VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Webový server v obvodu typu FPGA
Web Server in FPGA circuit

2011 Bc. Filip Frank

Prohlašuji, že jsem tuto diplomovou práci napsal samostatně. Uvedl jsem veškeré literární prameny a publikace ze kterých jsem čerpal

V Ostravě dne 2.5 ….....

Rád bych poděkoval všem, kteří mi poskytli cenné rady a připomínky při psaní této diplomové práce, především pak vedoucímu diplomové práce doc. Ing. Jaroslavu Zdrálkovi, Ph.D za počáteční informace a zapůjčení vývojové desky.

Abstrakt:

Diplomová práce se zabývá implementaci webového serveru do obvodu typu FPGA. Cílem je ukázková realizace na vývojové desce. Realizace využívá hradlové pole Xilinx Spartan3E, základ hardware systému tvoří softwarový mikroprocesor Microblaze. Celý návrh je řešen jako SoC vestavěný systém. Pro běh software slouží operační systém Linux 2.6, a je využito aplikace BusyBox. Systém je schopen komunikace přes metalický Ethernet 10/100M a umožňuje připojení pomocí protokolů IPv4 a IPv6. Jako ukázkový webový obsah jsou použity statické webové stránky. Systém lze také spravovat vzdáleně pomocí protokolu Telnet či SSH.

Klíčová slova:

FPGA, vestavěný systém, Linux, Microblaze, IP komponenta, BusyBox, SoC, MTD, Xilinx, webový server

Abstract:

This work is focused to implementation of web server to FPGA type circuit. Purpose is build sample implementation on development board. Implementation use gate array Xilinx Spartan3E, base of hardware is Microblaze soft-core IP microprocessor. All design is solved as SoC embedded system. Software is powered by Linux 2.6 operating system and using BusyBox application. System can communication over metallic Ethernet 10 or 100 megabit and support communication over IP protocol version 4 and 6. It using static HTML pages as sample for web server site. System can be remote managed over Telnet or SSH protocol.

Keywords:

FPGA, embedded system, Linux, Microblaze, IP core, SoC, MTD, Xilinx, BusyBox, web server

Seznam použitých zkratek a symbolů:

BIN	- Binary
BRAM	- Block RAM
BSB	- Base System Builder
CAN	- Controller Area Network
CPLD	- Complex Programmable Logic Device
DDR	- Dual Data Rate
EDK	- Embedded Development Kit
EEPROM	- Electrically Erasable Programmable Read-Only Memory
ELF	- Executable and Linkable Format
FPGA	- Field Programmable Gate Array
FSL	- Fast Simplex Link
FTP	- File Transfer Protocol
GNU	- GNU's Not Unix
HTTP	- Hyper Text Transfer Protocol
I2C	- Inter-Integrated Circuit
IO	- Input Output
IP	- Intellectual Property, Internet Protocol
IPv4	- Internet Protocol version 4
IPv6	- Internet Protocol version 6
JFFS2	- Journalling Flash File System version 2
LUT	- Lookup Table
MAC	- Media Access Controller
MII	- Media Independent Interface
MMU	- Memory Management Unit
MS	- Microsoft
MTD	- Memory Technology Device
NAND	- Not AND
OS	- Operating System
PC	- Persona Computer
PHY	- Physical
PLB	- Processor Logical Bus
PLD	- Programmable Logic Device
RAM	- Random Access Memory
RISC	- Reduced Instruction Set Computer
RTOS	- Real Time Operating System
SATA	- Serial ATA
SDK	- Software Development Kit
SoC	- System on Chip
SPI	- Serial Peripheral Interface
SREC	- S-record format
SSH	- Secure Shell
TCL	- Tool Command Language
VHDL	- VHSIC hardware description language
XCL	- Xilinx Cache Link
XMD	- Xilinx Microprocessor Debugger
XPS	- Xilinx Platform Studio

Obsah

1. Úvod

V dnešní době se lze stále častěji setkat s obvody typu FPGA. Jejich nejznámější aplikací je vytváření vlastních návrhů logických obvodů, které se nevyplatí vyrábět sériově. Jejich výhodou je také snadný vývoj, neboť zapojení a tím i chování obvodu lze změnit pouze změnou obsahu jeho paměti s návrhem zapojení. Méně známým je pak využití těchto obvodů k vytváření takzvaných vestavěných systémů. Takovýto vestavěný systém je malým počítačem určeným pro montáž (vestavbu) do různých, obvykle jedno účelových zařízení. Takovými zařízeními jsou například směrovače, IP kamery, WiFi přístupové body, nápojový automat nebo chytré mobilní telefony.

Tyto systémy jsou konstruovány obvykle jako SoC systémy, tedy vše je umístěno v jednom integrovaném obvodu s minimem okolních součástek. Nejčastěji se dnes setkáváme s obvody, které jsou sériově vyráběné a nelze měnit jejich zapojení. Toto je možné právě u obvodů FPGA, a lze tedy i v hotovém výrobku doplnit zapojení a například využít volných IO pinů k přidání sběrnice I2C, SPI, CAN či SATA. Také je možno samozřejmě opravovat chyby v zapojení a to dokonce i za běhu celého systému. Ačkoli je hlavním návrhovým prvkem FPGA obvodů jazyk VHDL či Verilog, u těchto systémů se používají k návrhu již hotové komponenty, které následně jen konfigurujeme a skládáme.

Tato diplomová se zabývá tvorbou hardwarového i softwarového návrhu, který bude plnit funkci Webového serveru komunikujícího přes rozhraní Ethernet. K tomu bude použito vývojové desky s FPGA obvodem. Cílem není vytvoření nějakého smysluplně fungujícího zařízení, ale jedná se především o nalezení postupu jak takový systém vytvořit. A samotný webový server nám bude sloužit pouze jako ukázka použití a funkčnosti. První část práce se zbývá základními teoretickými poznatky pro návrh takového systému, zde patří i popis vývojového prostředí a možností návrhu aplikací. Dále se pak zabývá postupem vytvoření hardwarového a softwarového návrhu, tato část je psána tak aby opakováním těchto kroků bylo možno sestavit stejný systém jako jsem vytvořil já.

V závěru se pak věnuji zhodnocení dosaženého návrhu a zabývám se možnostmi využití a navázání na tuto práci, neboť možnosti těchto systémů díky použití FPGA obvodů daleko překračují rámec této práce.

V některých částech této práce jsou použity anglické výrazy, které sice mají český překlad avšak mnohdy s neodpovídajícím významem. Všechny tyto výrazy jsou při svém prvním výskytu v textu vysvětleny českým textem. K porozumění obsahu práce je dobré znát základní principy fungování počítačů, operačního sytému Linux a kompilace zdrojových kódů aplikací.

2. Vestavěné Systémy

Je to označení pro systémy, kde řídící počítač je vestavěn do zařízení jako celku. Jde například o bankomaty, PDA, tiskárny, mp3 přehrávače, mobilní telefony, routery, skenery. Na rozdíl od běžného počítače jsou tedy určeny k jednoúčelové činnosti. Typickým řešením takového systému je umístění mikroprocesoru spolu s další logikou zařízení na jednu desku plošných spojů. Pro vestavěné systémy také existují specializované operační systémy. Nejznámějším je RTOS (Real Time Operating systém – systém pro práci v reálném čase), uCLinux (pro jednoduší zařízení/mikroprocesory), Linux nebo i speciální edice MS Windows. Tento software řídící vestavěný systém obvykle nazýváme firmware a dnes bývá nejčastěji umístěn v paměti typu Flash. Pro tyto systémy je také typické, že jsou velmi úsporné velikostí pamětí, zatím co dnes máme v počítačích gigabajty úložné i operační paměti, vestavěné systémy mají obvykle jen jednotky či desítky megabajtů. Proto jsou pro tyto systémy vyvíjeny nejen speciální operační systémy, ale také úsporné aplikace. V dnešní době miniaturizace, se stále více prosazuje systém SoC, který se snaží co nejvíce potřebných součástí umístit pouze do jednoho čipu. To vede k další miniaturizaci vestavěných systémů a snížení výrobních nákladů. Principu vestavěného systému s SoC je využito i v této práci.

2.1 SoC systémy

Označením SoC rozumíme systémy, kde je návrh umístěn do jednoho integrovaného obvodu-čipu s minimem externích součástek. V takovém integrovaném obvodu je pak například implementován mikroprocesor, sběrnice, řadič RS-232, Ethernetový kontrolér, I2C rozhraní, a také paměti typu Flash či RAM. V dnešní době je takovým typickým příkladem třeba MP3 přehrávač, kde mikroprocesor, paměť, řadič rozhraní USB, řadič displeje, ovládání tlačítek, DA převodník i koncový stupeň implementován pouze v jediném integrovaném obvodu. Kromě výhody v podobě miniaturizace a tím nízké ceny, je třeba zmínit také nevýhody. Například u MP3 přehrávače, či SoC zvukových karet se může negativně projevit rušení, které se indukuje v analogové části obvodu od části digitální. I to však výrobci řeší a neustále vylepšují lepším řešením desek plošných spojů či různým napájením pro obě části obvodu. Obvody tohoto SoC typu se vyrábějí jednak na zakázku, což je vhodné pro velké výrobní série, nebo se jedná o obvody typu programovatelných logických prvků, dnes především CPLD či FPGA. Tyto prvky se využívají nejen pro finální návrh, ale také díky své snadné rekonfigurovatelnosti i pro vývoj zákaznických obvodů. Těmto typům obvodů se věnuje následující kapitola.

2.3 Programovatelné logické prvky

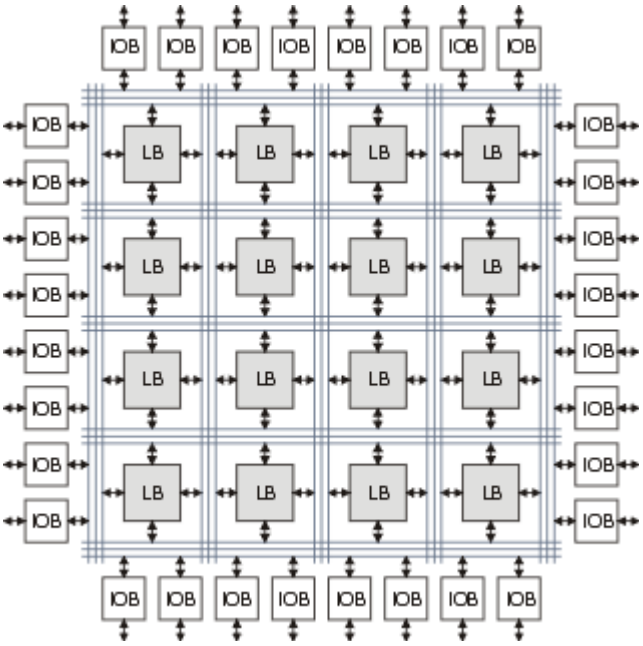
Jsou to logické obvody, které obsahují množství univerzálních hradel, ze kterých je jejich propojením možné vytvořit téměř libovolný logický obvod a kdykoli toto zapojení měnit. Tyto obvody označujeme zkratkou PLD (Programmable Logic Device). Programovatelná hradlová pole jsou velmi populární konstrukční alternativou k zákaznickým integrovaným obvodům. Výhodou je jejich možnosti přeprogramování, a u některých i za běhu. Nevýhodou pak je nižší použitelná taktovací frekvence na rozdíl od jednoúčelových obvodů. Lze je rozdělit do tří skupin

- klasické PLD
- komplexní PLD (CPLD)
- FPGA

Klasické obvody PLD obsahují jen pole hradel například typu AND-OR propojené pomocí matice. Umožňují realizovat jen jednoduché funkce. Jsou konstrukčně nejstarší a dnes se již téměř nepoužívají. Obvody CPLD pak shlukují více klasických PLD do buněk, a teprve tyto buňky jsou připojeny ke globální propojovací matici. Tyto obvody také obsahují interní EEPROM paměť pro zápis konfigurace. Dnes se tyto obvody používají pro jednodušší aplikace. FPGA obvody jsou pak nejsložitějšími a dnes nejpoužívanějšími obvody, a proto jim je věnována celá následující kapitola. Základní informace o PLD obvodech lze najít v literatuře [10].

2.4 Obvody FPGA

FPGA je zkratkou pro Field Programmable Gate Arrays. Mají nejobecnější strukturu ze všech typů programovatelných obvodů. Obvykle také obsahují nejvíce logiky (největší a současné FPGA obvody obsahují až miliony ekvivalentních hradel). Také obsahují například i jiné hotové bloky, jako třeba hardwarový mikroprocesor PowerPC či Intel Atom u hradlových polí Xilinx Virtex. Tyto obvody mají pro svou konfiguraci integrovánu paměť SRAM, které je závislá na napájení, a proto je u nich nutné zařídit nahrání návrhu při každém zapnutí. To se provádí připojením externí paměti, ze které si FPGA obvod umí sám konfiguraci přečíst. Na obrázku 1. je znázorněna základní struktura FPGA obvodu.



Obr. 1: Struktura FPGA obvodu

Bloky označené IOB (Input/Output Block) představují vstupně-výstupní obvody pro každý pin FPGA. Tyto bloky obvykle obsahují registr, budič, multiplexer a ochranné obvody. Bloky LB (Logic Block) představují vlastní programovatelné logické bloky. Tyto bloky jsou obvykle tvořeny LUT, sčítačkou a klopným obvodem typu D (registr). LUT umožňují vytvořit libovolnou kombinační logickou funkci. Všechny bloky mohou být různě propojeny globální propojovací maticí. FPGA obvykle umožňují propojit některé signály logických bloků přímo se sousedním bez nutnosti využívat globální propojovací matici. Takovéto spoje mají mnohem menší zpoždění a umožňují tak realizovat například rychlé obvody šíření přenosu, což je nezbytné pro sčítačky nebo násobičky.

Kromě bloků znázorněných na předchozích obrázcích integrují výrobci do FPGA další prvky. Většina moderních FPGA obsahuje několik bloků rychlé synchronní statické paměti RAM. Velmi často obvody FPGA obsahují PLL (Phase Locked Loop) nebo DLL (Delay Locked Loop) pro obnovení charakteristik hodinového signálu, případně pro násobení nebo dělení jeho frekvence.

Nejznámějšími výrobci těchto polí jsou Xilinx, Altera, Lattice nebo Actel. V této práci bude využito FPGA obvodu Spartan-3E XC3S500E od firmy Xilinx.

2.5 Popis Spartan-3E XCS500E

Všechny FPGA Xilinx používají konfiguraci pomocí statické paměti RAM. To znamená, že po připojení napájení je nutné vždy nahrát znovu konfiguraci. Výhodou tohoto řešení je téměř nekonečná reprogramovatelnost FPGA a také rychlost. Tento obvod patří k nejnižší řadě FPGA obvodů firmy Xilinx.

Vlastnosti:

- 500 tisíc hradel
- 360k bit blokové RAM (Bloková paměť RAM je tvořena skutečnými bloky synchronní statické RAM. Bloky mají dvojnásobné adresové i datové sběrnice, takže je lze využít i jako dvouportovou paměť.)
- 20 násobiček
- 232 uživatelských IO pinů
- podpora DDR
- diferenciální vstupy/výstupy
- JTAG rozhraní
- umožňuje implementaci mikroprocesorů MicroBlaze a PicoBlaze

Vstupně-výstupní buňka může pracovat v jednom ze šestnácti režimů, jako jsou například LVCMOS, HSTTL, STTL, GTL a další. Výstupy z těchto buněk lze propojit na kterýkoli pin pouzdra s výjimkou malého okruhu speciálních pinů. Vstupně/výstupním pinům lze také zařadit třeba pull-up, pull-down rezistory nebo terminátory sběrnic. Tento obvod se také vyrábí v pouzdrech s různým provedením. Pro tuto práci bude použita BGA varianta čipu na vývojové desce Xilinx SPARTAN-3E Starter board.

Tento obvod umožňuje ukládat svou konfigurací do pamětí Xilinx Platform Flash, Flash a SPI. Konfiguraci lze také nahrávat pomocí ladícího JTAG rozhraní. V této práci je použita Platform Flash. Tento obvod také umožňuje mít obsah paměti chráněn šifrováním proti krádeži návrhu. U paměti Flash jsou k dispozici dvě možnosti. První z nich je umístění konfigurace na počátek této paměti (adresa 0x0), a obvod svůj obsah čte po adresách směrem nahoru. Druhou možností je umístění na konec této paměti a obvod pak čte návrh po paměťových adresách směrem dolů. Pokud se využívá těchto možností umístění je potřeba dát pozor, aby nedošlo k případné kolizi s ostatním návrhem umístěným paměti.

2.6 SPARTAN-3E Starter board

Je to vývojová deska s hradlovým polem Spartan-3E XCS500E, a dalšími periferiemi nutnými pro vývoj vestavěných systémů s FPGA obvodem.

Deska obsahuje následující prvky:

Xilinx obvody:

- Spartan-3E FPGA (XC3S500E-4FG320C)
- CoolRunner™-II CPLD (XC2C64A-5VQ44C)
- Platform Flash (XCF04S-VO20C)
- Clocks: 50 MHz krystalový oscilátor

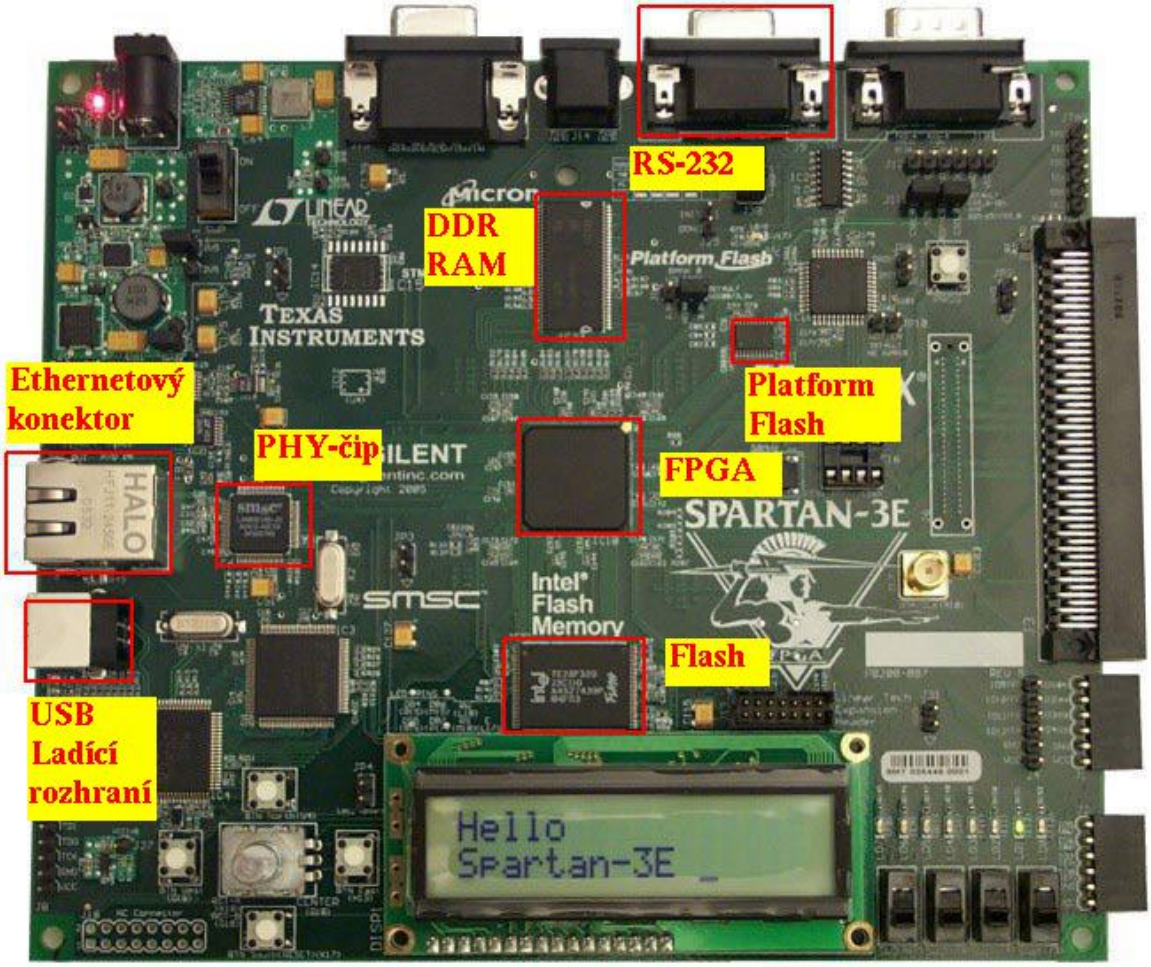
Paměti:

- 16 MB Parallel Flash
- 2 MB SPI Flash
- 64 MB DDR SDRAM

Rozhraní a konektory:

- Ethernet 10/100 PHY-chip(SMC LAN83C185) +
- JTAG USB rozhraní
- 2x 9-pin RS-232 sériový port (DTE+DCE)
- PS/2- port pro PC myš nebo klávesnici
- Skokový nekonečný potenciometr s tlačítky
- 4x posuvný přepínač
- 8xLED diody
- 100-pinový rozšiřující konektor
- 3x šestipinový rozšiřující konektor
- Dvouřádkový znakový LCD displej
- VGA port pro připojení PC monitoru
- DA/AD převodníky
- SMA konektor pro přivedení externího zdroje hodinového signálu

Na obrázku 2. je fotografie desky, červeně jsou vyznačeny části, které budou využity pro realizaci této práce. Jedná se samotný FPGA prvek (uprostřed), PHY-čip, Ethernetový konektor, RS-232, paměť RAM, paměť Flash, Xilinx Platform Flash a ladící port. Podrobnější informace o vývojové desce nalezneme v jejím uživatelském manuálu [1].



Obr. 2: Vývojová deska

2.7 IP komponenty

IP (Intellectual Property) jsou komponenty napsané v jazyku VHDL či Verilog, které implementují nějakou komponentu. Takovou komponentou může být například mikroprocesor, řadič paměti, sběrnice, Ethernetový kontrolér či grafický čip. Tyto komponenty velmi usnadňují vývoj, kdy místo psaní vlastní již dávno vymyšlených součásti můžeme využít již existující komponenty a skládat je dohromady ve složitější systém či propojit s komponentami vlastními. Výhodou těchto komponent je především jejich možná konfigurovatelnost a nezávislost na cílové platformě, neboť se jedná jen o popis funkce. Konkrétní zapojení se vytvoří až syntézou v programu obvykle dodaným výrobcem obvodu (lze však použít i jiné programy např. od fy. Mentor) do kterého návrh umisťujeme. Tyto komponenty dodává buď výrobce v rámci svého

vývojového prostředí, lze je nakupovat, nebo jsou k dispozici zdarma například na Internetu. Součástí komponent bývají také nezbytné ovladače pro operační systém či knihovny pro vývoj vlastních aplikací s těmito komponentami. Na IP komponentách, dodaných s vývojovým prostředím výrobce, je postavena i realizace v této práci. Základní použitou IP komponentou je softwarový mikroprocesor Microblaze

2.8 Softwarový mikroprocesor

Softwarový mikroprocesor je označení pro mikroprocesory implementované pomocí IP komponenty. Nejedná se tedy o procesor jako součástku, kterou by jsme si mohli koupit, ale pouze o kód popisující jeho chování. Tento kód je napsán v jazyce VHDL nebo Verilog a z tohoto kódu pak specializovaný software provede návrh zapojení hradel pro hradlové pole (propojovací matice). Nejedná se tedy o nějakou emulaci procesoru, jak by se z názvu mohlo zdát. Výhodou softwarových mikroprocesorů je jejich konfigurovatelnost, lze u nich zařadit či nezařadit rozšiřující prvky (například ladící rozhraní či funkci zpracování výjimek). A to pouhou změnou jejich kódu (jsou-li volně dostupné) nebo častěji pomocí konfiguračního souboru. Tuto konfiguraci lze také provádět v grafickém vývojovém prostředí. Nevýhodou těchto procesorů je pomalejší realizace v FPGA prvcích, co se dosažené taktovací frekvence týče, což však nemusí nutně znamenat malý výkon. Používají se především ve vestavěných systémech právě s FPGA prvky, kdy lze mít na jednom čipu hradlového pole jak mikroprocesor, tak i periferie jako VGA adaptér, Ethernet adaptér, řadič RS232, či jinou logiku.

Nejznámějšími softwarovými mikroprocesory jsou:

- MicroBlaze - Xilinx
- PicoBlaze - Xilinx
- OpenSPARC T1 - Sun
- OpenRISC - Opencores.org
- OpenFire
- TSK3000A

Některé z nich mají otevřené zdrojové kódy například OpenRISC z projektu www.opencores.org, kde lze nalézt i mnoho jiných softwarových mikroprocesorů.

2.9 Mikroprocesor Microblaze

Microblaze je softwarový mikroprocesor vyvinutý firmou Xilinx pro obvody typu FPGA. Instrukční sada tohoto dvaatřicetibitového mikroprocesoru ze založena na architektuře RISC, většinu instrukcí mikroprocesor zpracuje v jednom cyklu. Mikroprocesor pro připojení externích zařízení používá sběrnici PLB s řízením typu master/slave. Pro přístup k operační paměti však používá sběrnici XCL. Procesor obsahuje dvaatřicet třiceti dvoubitových registrů.

Některé vlastnosti mikroprocesoru je možné uživatelsky nakonfigurovat:

- pipeline 3 nebo 5ti stupňová
- velikost instrukčních i datových vyrovnávacích (cache) pamětí
- podpora MMU
- taktovací frekvence
- ladící (debug) rozhraní
- možnost dvou jádrové konfigurace
- podpora násobičky/děličky
- obsluha výjimek
- optimalizace
- přerušovací systém

Mikroprocesor umožňuje připojit pomocí FSL sběrnice coprocesor, ve kterém lze například posloupnost často využívaných instrukcí aplikace realizovat hardwarově, a tím velmi výrazně zrychlit vykonávání kódu. Lze tak tedy vytvořit vlastní uživatelské instrukce.

Microblaze bez jednotky MMU umožňuje například běh operačních systémů FreeRTOS nebo uCLinux. S MMU pak i operační systémy vyžadující hardwarovou správu stránkování a ochrany operační paměti. Typickým příkladem je operační systém Linux. Od roku 2009 je mikroprocesor podporován jako první softwarový přímo jeho hlavní vývojovou řadou (mainline), která se nachází na Internetových stránkách www.kernel.org. Mnoho dalších informací a přesný popis mikroprocesoru lze nalézt v literatuře [5].

2.10 Možnosti návrhu software

Vestavěné systémy nejsou specifické jen po stránce hardwarové, ale i softwarové. Používají se odlišné typy operačních systémů a provozovaných aplikací, je to způsobeno především nutností vypořádat se s malými velikostmi pamětí, a také pomalejšími procesory. Z toho důvodu se používají jednodušší implementace aplikací. Pro vystavěné systémy s mikroprocesory Microblaze máme na výběr ze tří možností jak navrhnout software.

- Samostatný návrh (Standalone)
- XilKernel
- Plnohodnotný operační systém

První možností je samostatný systém, který je určen pro jednodušší návrhy. Pro vývoj takovéto aplikace jsou k dispozici připravené základní funkce jazyka C, které jsou určeny k zjednodušení vývoje a psaní kódu. Obsahuje známé funkce jako je memcpy, printf, či funkce pro inicializaci, záchyt výjimek a přerušení mikroprocesoru. Seznam a popis těchto funkcí je v literatuře [4]. Této možnosti samostatného návrhu je v této práci využito pro zavaděč operačního systému.

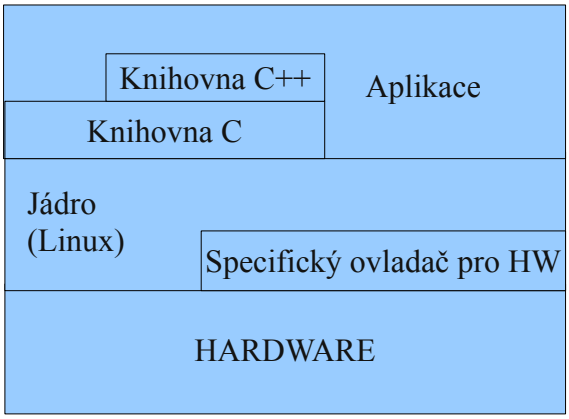
Druhou možností je XilKernel, jedná o jádro operačního systému, které nabízí především funkce pro realizaci více vláknových aplikací. Nabízí synchronizační služby, fronty zpráv, sdílenou paměť, podporu vláken s prioritním plánováním, či softwarové časovače.

Poslední možnost je dnes pravděpodobně nejvíce užívaná, pokud nemáme zvláštní nároky na rychlost či minimalizování využití paměti. Plnohodnotným operačním systémem je například Linux, uCLinux či RTOS. Takový systém obsahuje vše potřebné, jako jsou ovladače, komunikaci přes IP, správu paměti nebo síťování. Hardwarové rozdíly jsou zcela odstíněny od běžících aplikací. Pro realizaci v této práce bude využito operačního systému Linux verze 2.6.

Jako aplikace, která bude reprezentovat webový server byl zvolen Busybox. Ten je dnes nejznámější aplikací používanou ve vestavěných systémech s operačními systémy Linux. Implementuje mimo již zmíněného webového serveru, téměř kompletní základ celého Linuxového systému, tak jak jej známe například z počítačů PC.

Kromě samotné aplikace potřebujeme ještě základní knihovnu C. Ta je již součástí GNU toolchain a bude použita pro tuto realizaci, její vlastní vytvoření/kompilace je sice možné, ale složité, zdlouhavé a vyžaduje mnoho různých oprav.

Struktura systému s OS Linux:



Obr. 3: Struktura systému s OS Linux

Na obrázku 3. je znázorněna struktura systému s OS Linux. Základem je Hardware, na kterém bude systém běžet, nad ním pak běží operační systém. Samotné jádro běží přímo na mikroprocesoru, avšak pro ostatní hardware musí být použito speciálních ovladačů pro jejich ovládání, patří sem také firmware pro tyto zařízení. Jádro pak přebírá ke zpracování funkce ze standardní knihovny C, která přebírá funkce od jiných knihoven či přímo od aplikace. Také sama aplikace může přímo komunikovat s jádrem, toho se však používá pouze pro speciální účely a snažíme se tomuto vyhnout. Předmětem této části práce bude nakonfigurovat linuxové jádro a připravit celý systém až po cílovou aplikaci k běhu na vývojové desce.

3 Vývojové prostředí

K vývoji bylo využito vývojové prostředí Xilinx ISE Design Suite, které dodává firma Xilinx. Jedná se o rozsáhlý balík nástrojů pro návrh aplikací všech typů hradlových polí firmy Xilinx. Bylo použito verze 12.4 pro operační systém Linux. Celé prostředí bylo zprovozněno pod operačním systémem Gentoo Linux 64 bit. Zprovoznění prostředí na tomto systému Linux však není úplně triviální a skýtá mnoho nečekaných problémů. V příloze 1. je proto návod na zprovoznění nad rámec instalační příručky neboť bez tohoto kroku nelze pokračovat a prostředí na tomto systému není plně funkční. Firma Xilinx podporuje oficiálně pouze operační systémy Red Hat Linux či Windows, které jsou však placené.

3.1 Xilinx EDK

EDK je vývojový balík aplikací, který je součástí ISE a je učen k vytváření vestavěných systémů s mikroprocesory PicoBlaze, MicroBlaze a PowerPC, které jsou postaveny na hradlových polích typu FPGA firmy Xilinx. Sestává se ze dvou hlavních aplikací XPS a SDK. XPS slouží k návrhu hardwarové struktury vestavěného systému (procesor, pamětí, I/O periferie, sběrnice, adresy,..). SDK pak k návrhu aplikací pro hardware vytvořený v XPS. Podrobný popis všech součástí EDK prostředí nalezneme v literatuře [2]. V práci se také okrajově využívá ladící program (debugger) XMD a nástroj Impact.

3.2 Xilinx XPS

Prostředí XPS slouží k návrhu hardwarové části projektu. K dispozici jsou IP komponenty, které lze přidávat do návrhu, navrhnout jejich propojení, konfiguraci, nastavení jejich adres. Toto prostředí také obsahuje informace o vývojových deskách Xilinx, a to například použité hradlové pole, zapojení vstupně/výstupních pinů a externích komponent. Toho využívá generátor systému obsažený v XPS, který umožňuje na základě těchto znalostí vytvořit základ systému bez nutnosti zadávat například přesné zapojení prvků na vývojové desce, především tedy jejich připojení k FPGA obvodu. Dále prostředí umí tento návrh převést na soubor popisující zapojení na čipu. Tento soubor se nazývá bitstream. Prostředí také umí generovat různé informace například o využití prvků na čipu, či blokové zapojení. A samozřejmě umí umístit vytvořený návrh do obvodu.

3.3 Xilinx SDK

SDK slouží jako vývojové prostředí pro aplikace, které budou na našem vestavěném systému běžet. Umožňuje především psaní, kompilaci a ladění kódu, dále obsahuje různé knihovny a aplikace, které lze využít k zjednodušení a rychlejšímu návrhu. SDK je postaveno na opensource vývojovém prostředí Eclipse, a GNU Toolchain (GNU kompilátor, GNU debugger). Umožňuje také zápis do Flash paměti či nahrávání zapojení do obvodu. V tomto projektu je SDK prostředí použito pouze pro vývoj zavaděče jádra a nahrávání obsahu do paměti Flash.

3.5 GNU ToolChain

GNU toolchain je sada nástrojů a knihoven pro kompilaci programů pro architekturu Microblaze, jedná se například o kompilátor, linker. Samotné kompilační programy jsou určeny pro PC architekturu x86, ale generují kód pro architekturu Microblaze. Proto se tento typ nástrojů nazývá nástroji pro křížovou kompilaci. Pro své mikroprocesory Microblaze jej firma Xilinx přímo poskytuje. Je nutné jej pouze následujícím způsobem správně zprovoznit.

Postup:

1. Stáhneme pomocí `git clone git://git.xilinx.com/xldk/microblaze_v1.0.git`
2. Přejmenujeme microblaze_v1.0 na GNU_toolchain
3. rozbalíme pomocí `tar jxpf mb_gnu_tools_bin.tar.bz` a následně přejmenujeme složku microblaze-unknown-linux-gnu na mb_cross_compiler
4. Následně je nutné zmenit systémové proměnné PATH tak aby obsahovala cestu ke kompilátoru: např. `export PATH=/home/uzivatel/Xilinx/FPGA_Web/GNU_toolchain/microblaze-unknown-linux-gnu/bin:$PATH`
5. Nastavení proměnné pro křížovou kompilaci: `export CROSS_COMPILE=mb-linux-`

Poslední dva body nastavení proměnného prostředí jsou platné vždy jen pro aktuální shell, po jeho ukončení je nutné tyto proměnné nastavit znovu před užitím nástrojů. K tomuto účelu vytvoříme skript `cross_setup.sh` v hlavním adresáři projektu s následujícím obsahem:

```
adresar=`pwd`
PATH=$adresar/GNU_toolchain/mb_cross_compiler/bin:$PATH
export CROSS_COMPILE=mb-linux-
```

Tento skript pak vždy před první křížovou kompilací v shellu spustíme příkazem:
`source cross_setup.sh`

Nyní již máme vše potřebné pro kompilaci Linuxového jádra a aplikací.

4. Návrh Hardware

Vzhledem k tomu, že hradlová pole obsahuje pouze univerzální prvky, které jsou schopny plnit smysluplné funkce teprve až po jejich propojení je nutné navrhnout a vytvořit toto projení. Obvykle je u obvodů typu hradlové pole využito návrhu zapojení pomocí jazyka VHDL či Verilog. Ovšem pro takovéto složitější návrhy se používají již hotové komponenty, které připravil výrobce či jsou dostupné na Internetu (např. www.opencores.org). Tyto komponenty nazýváme IP a cílem návrhu je jich správné propojení a konfigurace i s ohledem na později provozovaný operační systém. Tímto si na FPGA prvku vytváříme naprosto vlastní a originální konfiguraci hardware. Celý návrh, kromě finálního umístění do paměti Platform Flash, je vytvářen v aplikaci Xilinx XPS.

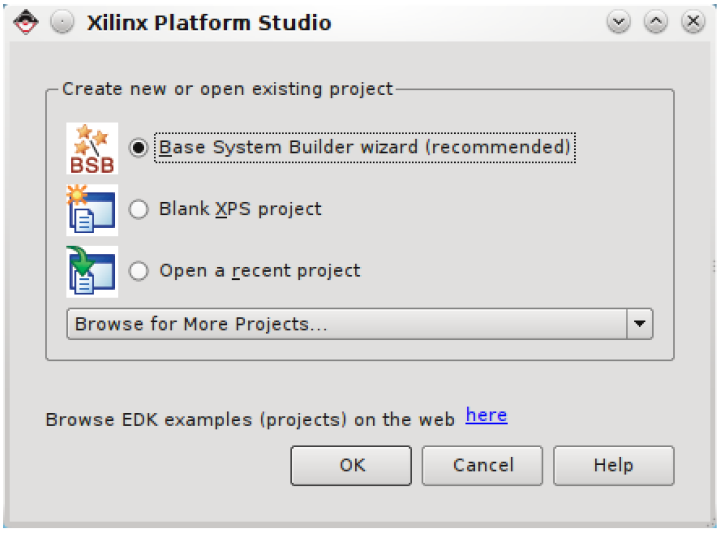
Základním prvkem je mikroprocesor Microblaze, dále se se budou využívat IP komponenty pro Ethernet kontrolér-MAC část, řadič paměti RAM, Flash a BRAM, časovače, řadič přerušení, sériový port, ladící modul, generátor hodin a resetu. Tyto použité IP komponenty dodává firma Xilinx jako součást EDK. Všechny tyto prvky budou umístěny v obvodu FPGA. Mimo FPGA obvod pak bude následující:

- Paměť DDR RAM
- Paměť Flash
- Převodník úrovní pro sériový port
- PHY-čip Ethernetového rozhraní
- Platform Flash s pomocným CPLD (není součástí návrhu, složí ke konfiguraci FPGA)

Následující postup vede k vytvoření požadovaného hardwarového návrhu. Prostředí XPS, s využitím BSB, který slouží vytvoření základu návrhu.

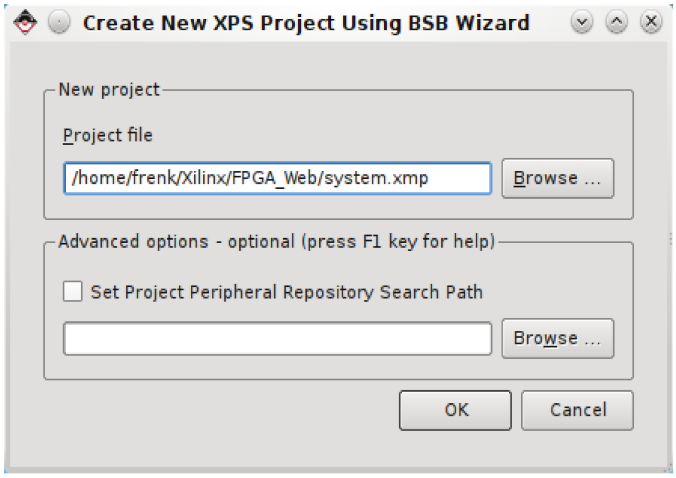
4.1 Vytvoření základního systému

Začneme spuštěním aplikace XPS, tak jak již bylo popsáno. Po spuštění se zobrazí okno (Obr. 4) s možnostmi BSB průvodce, vytvoření čistého projektu a otevření existujícího. Zvolíme tedy první možnost Base systém Builder wizard.



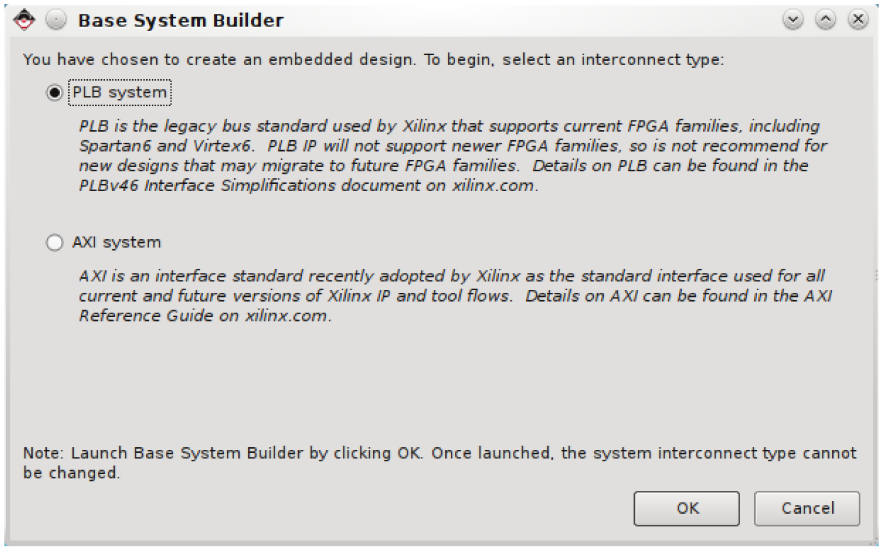
Obr. 4: Volba umístění projektu

Dalším krokem je výběr umístění projektu (Obr. 5), zde tlačítkem Browse u pole Project File otevřeme dialog a vybereme cestu a jméno projektu, nebo jej můžeme vyplnit manuálně přímo v textovém poli. V mém případě bylo použito /home/frenk/Xilinx/FPGA_Web/system.xmp. Vzhledem k tomu, že pracujeme v systému Linux je důležité brát zde ohled na velikost písmen. To se týká i všech dalších části projektu.



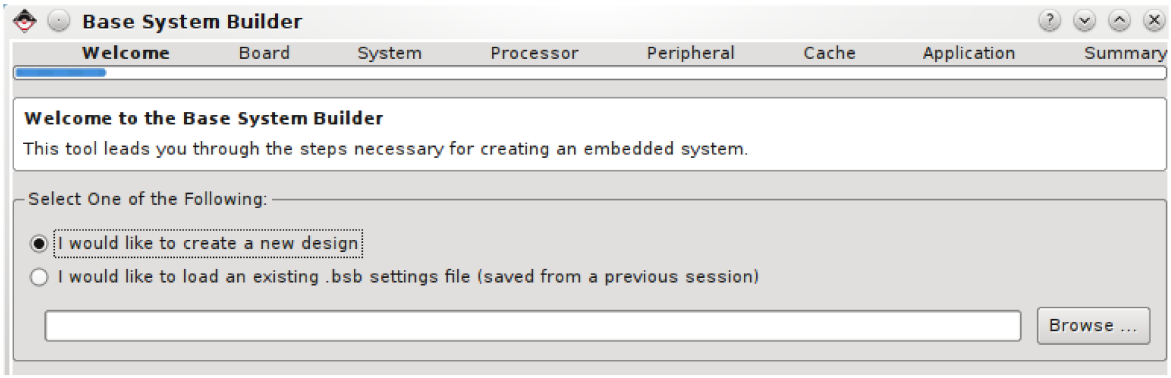
Obr. 5: Výběr umístění projektu

Dále pak je třeba zvolit typ návrhu dle používané sběrnice (Obr. 6), k dispozici je starší PLB a novější AXI. Vzhledem k tomu že novější AXI v době realizace této práce byla k dispozici pouze na nejnovějších polích Xilinx, volíme systém PLB.



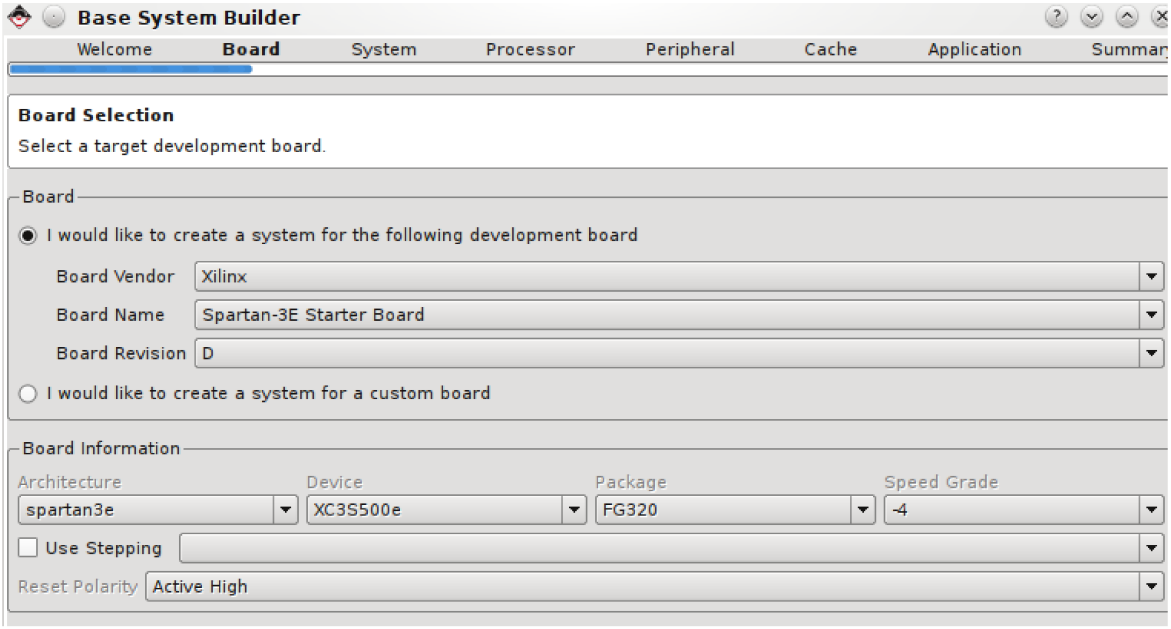
Obr. 6: Volba systémové sběrnice

Po výběru sběrnice se již zobrazí vlastní BSB průvodce (Obr. 7). V prvním jeho dialogu můžeme vybrat, zda vytváříme nový návrh nebo chce použít nějaký stávající. Zvolíme tedy vytvoření nového.



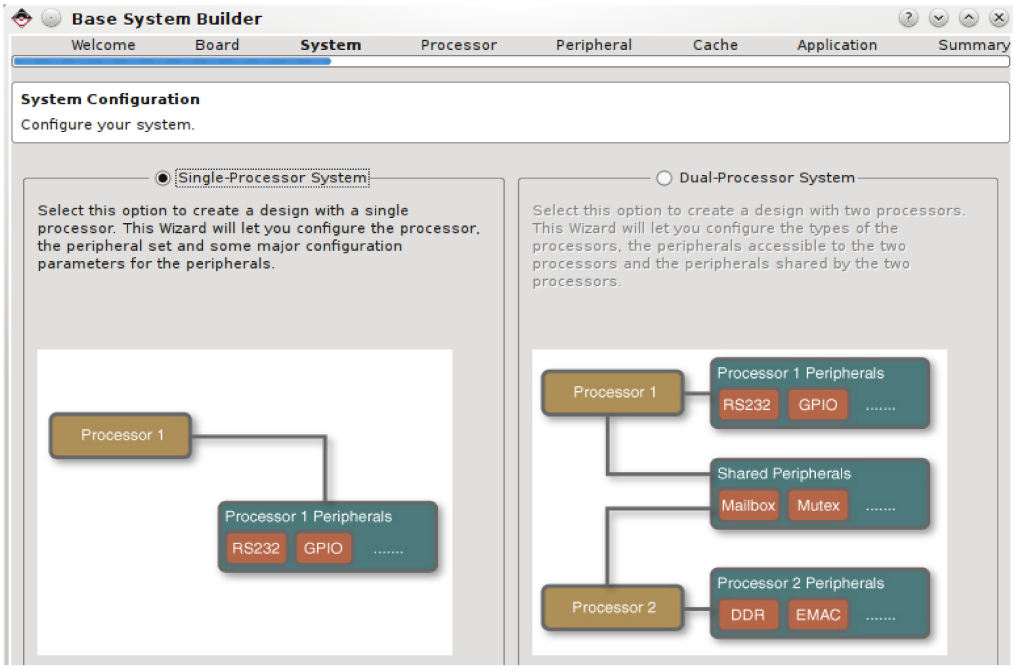
Obr. 7: Volba projektu

V dalším kroku (Obr. 8) vybereme používanou vývojovou desku. Zvolíme tedy výrobce Xilinx, název desky Spartan-3E Starter Board. Revize D. Níže pod výběrovým dialogem se zobrazí informace o používaném obvodu na zvolené desce. Konkrétně jeho architektura, typ, pouzdro a rychlostní skupina.



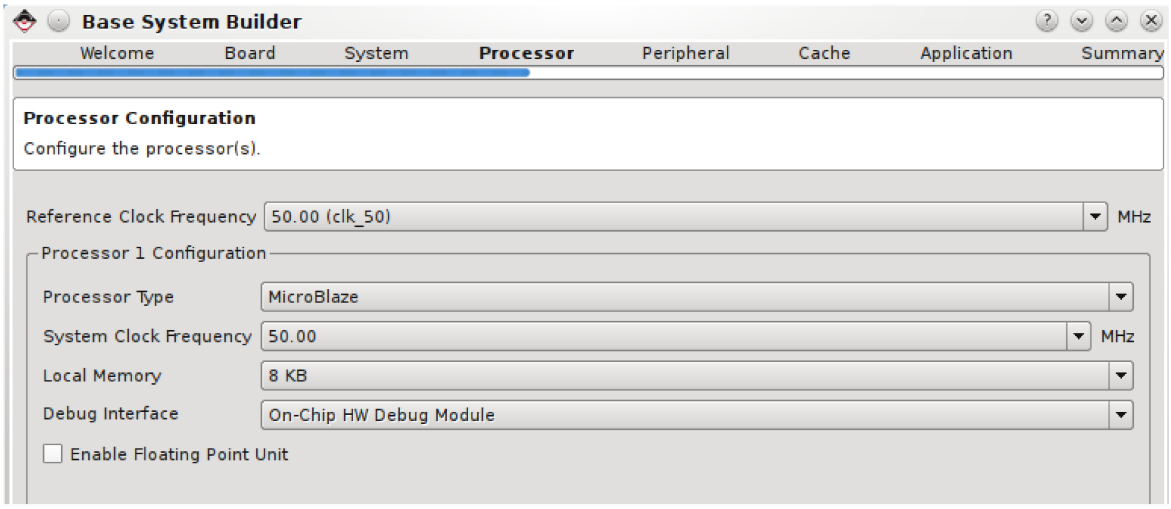
Obr. 8: Volba vývojové desky

Dále můžeme zvolit zda vytváříme jednoprocesorový či více procesorový systém. Zůstaneme u jednoprocesorového (Obr. 9).

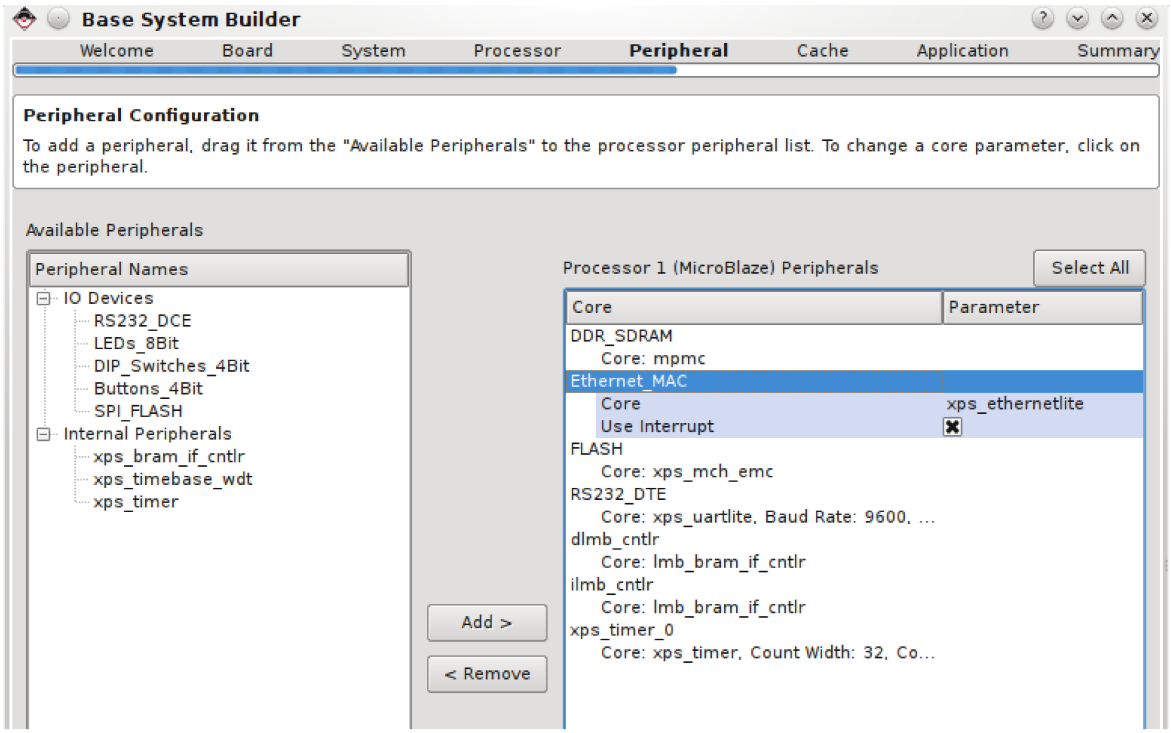


Obr. 9: Volba typu sytému

Dále následuje základní konfigurace mikroprocesoru (Obr. 10), zde si můžeme vybírat typy procesorů, avšak FPGA Spartan 3 podporuje pouze mikroprocesor Microblaze. Lze zde také zvolit frekvenci použitého oscilátoru, a frekvenci mikroprocesoru, velikost lokální paměti (využívá BRAM) či použití ladícího modulu. Předvyplněné hodnoty nám budou dostačovat, necháme tedy vše na výchozích hodnotách.



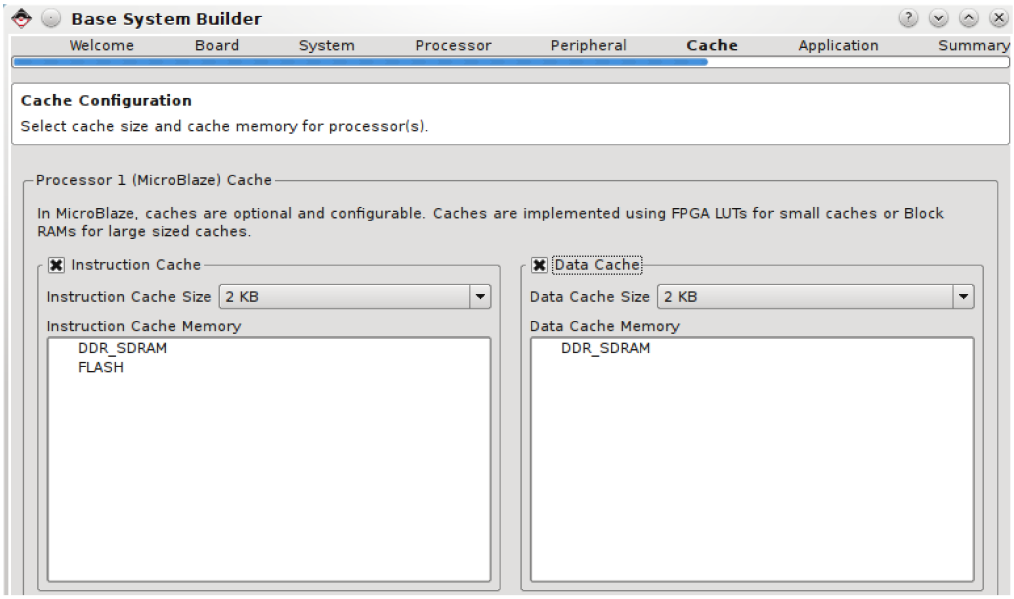
Obr. 10: Základní konfigurace mikroprocesoru



Obr. 11: Výběr komponent

Po tomto kroku následuje výběr komponent (Obr. 11). V levé části vidíme dostupné komponenty, v pravé přidané do našeho návrhu. Pro náš návrh je nutné odstranit komponenty Buttons_4Bit, RS232_DTE LEDs_8Bit a DIP_Switches neboť je nebudeme potřebovat a ušetříme tak jednak místo na čipu, tak i velmi významně čas generování bitstreamu. A Přidáme komponentu xps_timer_0, která je nezbytná pro provoz Linuxového jádra. Dále Linuxové jádro vyžaduje zapnout u komponent Ethernet MAC, RS232_DCE a xps_timer_0 přerušení, jako lze vidět na obrázku 11. u komponenty Ethernet_MAC.

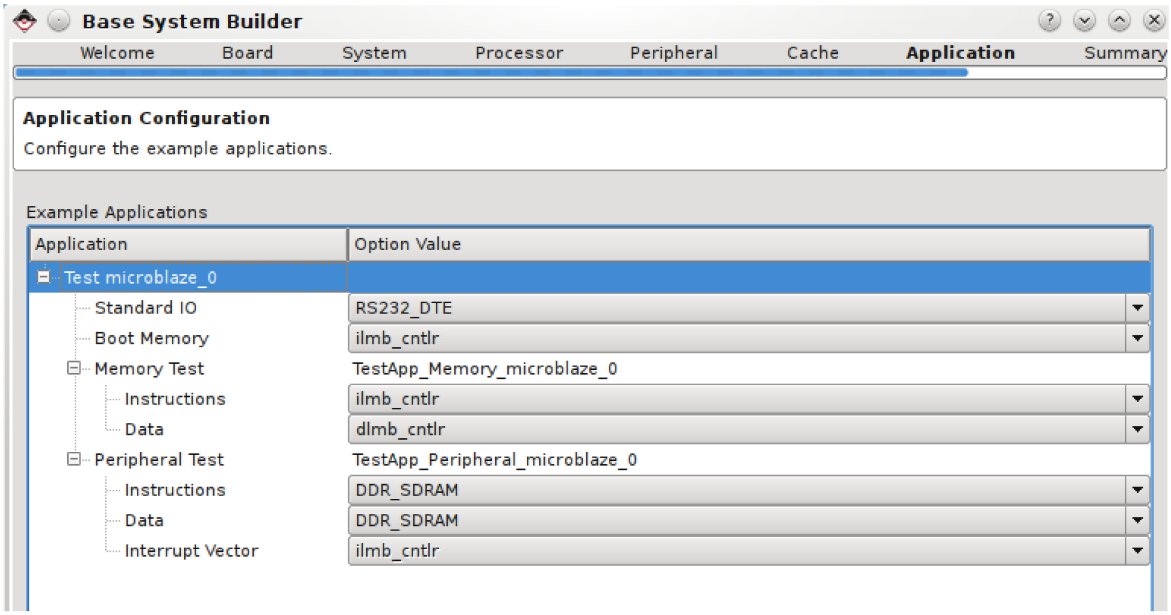
Dále lze zvolit obsazení pamětí vyrovnávacích pamětí cache (Obr. 12). Vzhledem k tomu, že paměti cache velmi přispívají k výkonu celého systému, povolíme je pro datovou i instrukční sběrnici o velikosti 2KB. Tyto pamětí využívají BRAM pro větší velikosti, pro menší LUT.



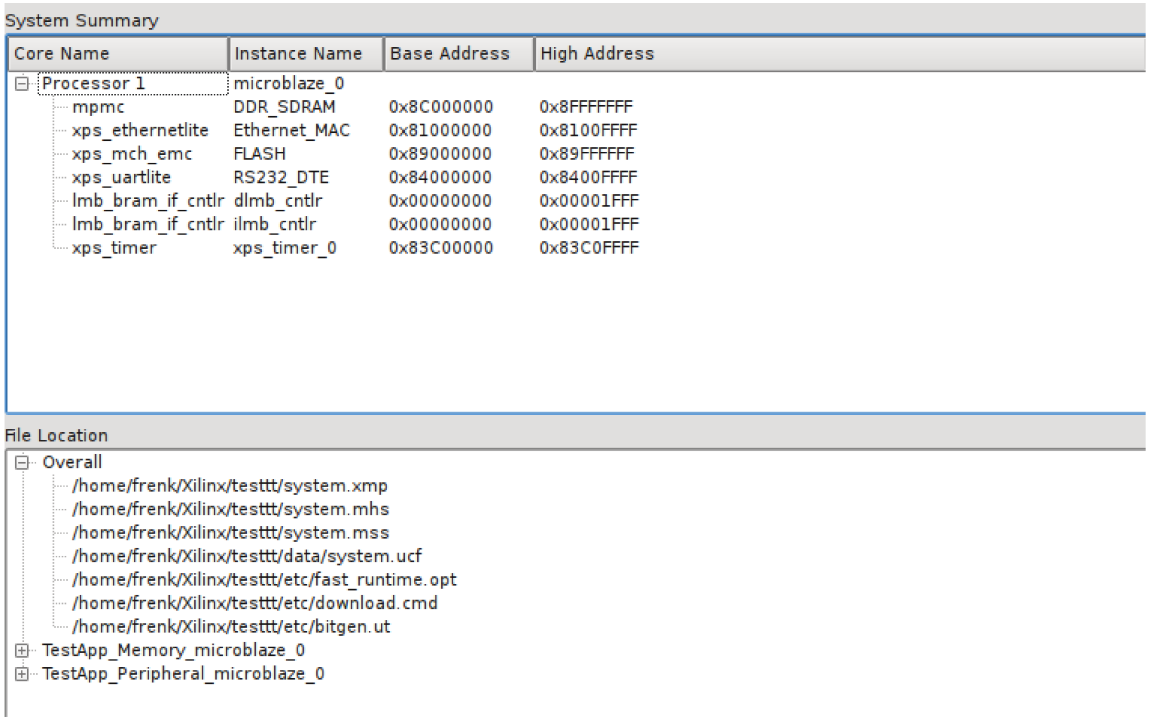
Obr. 12: Volba velikostí cache

V předposledním kroku lze zvolit nastavení zkušebních aplikací (Obr. 13). Tyto se hodí především na základní vyzkoušení funkčnosti návrhu. Lze zvolit jejich standardní výstup a umístění v pamětech. Zde vše ponecháme, neboť je nebude dále využívat. Jsou vhodné k vyzkoušení správné funkčnosti vývojové desky. Budu tedy dále předpokládat, že máme funkční nepoškozenou desku.

Dále pak již získáme jen informační stránku (Obr. 14), kde vidíme přehled o vytvořeném návrhu, používané komponenty, jejich IP název, základní a nejvyšší používané paměťové adresy. V dolní části pak je přehled o všech vygenerovaných souborech.



Obr. 13: Nastavení zkušebních aplikací

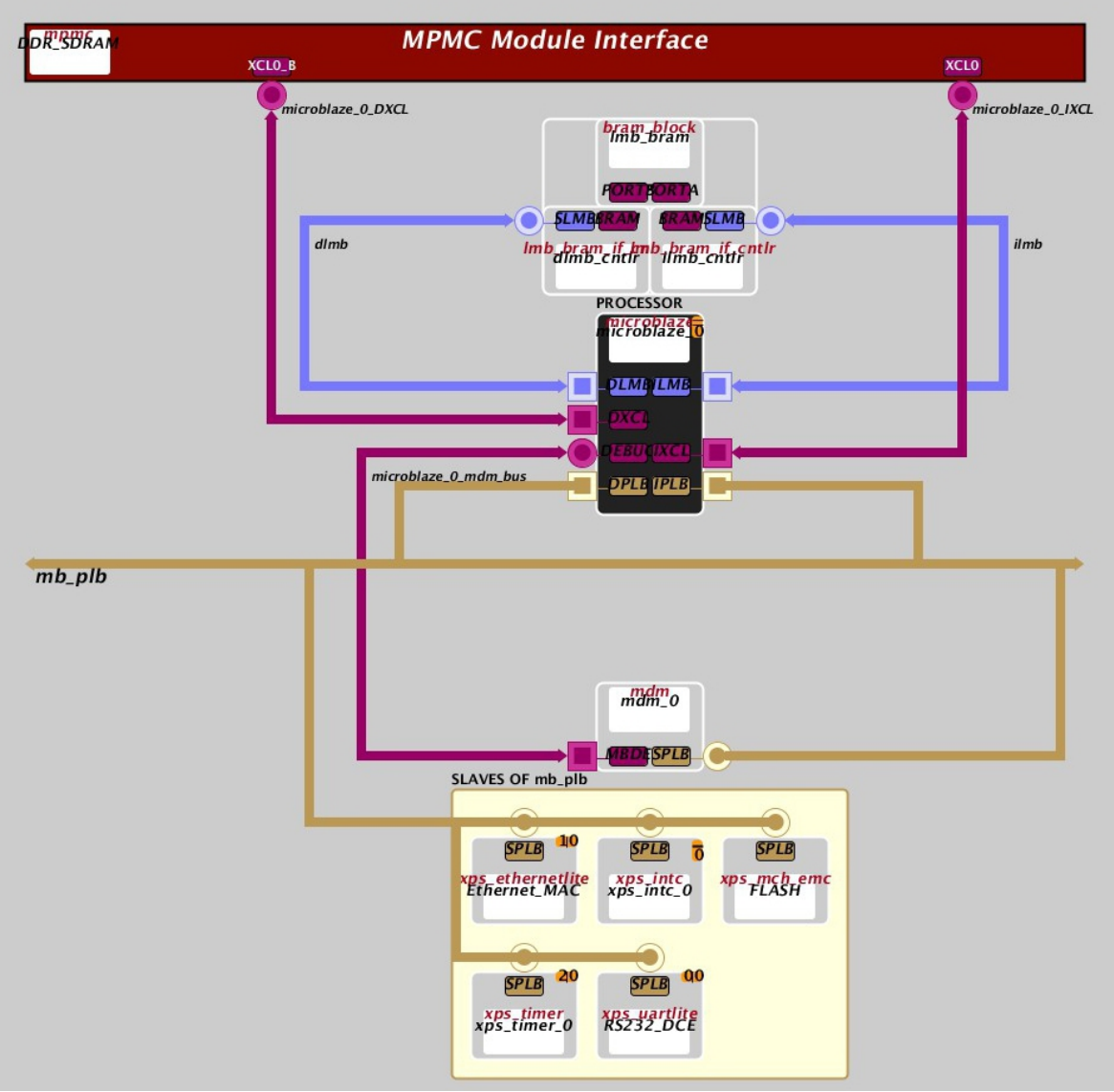


Obr. 14: Informace o systému

Po stisku tlačítka Finish se vytvoří projekt a následně se XPS zeptá na další krok

- Konfigurace ovladačů a knihoven
- Zápis návrhu do čipu a test
- Úprava aplikací vygenerovaných v BSB
- Používání XPS

Poslední volba je výchozí a vzhledem k tomu že potřebujeme ještě provést změny zvolíme tedy tuto možnost. Nyní je možné používat XPS a provádět změny ve vytvořeném návrhu. Na obrázku 15. můžeme vidět výsledné zapojení vygenerovaného návrhu, tak jak nám jej zobrazí XPS.



Obr. 15: Zapojení Hardwarového návrhu (export z XPS)

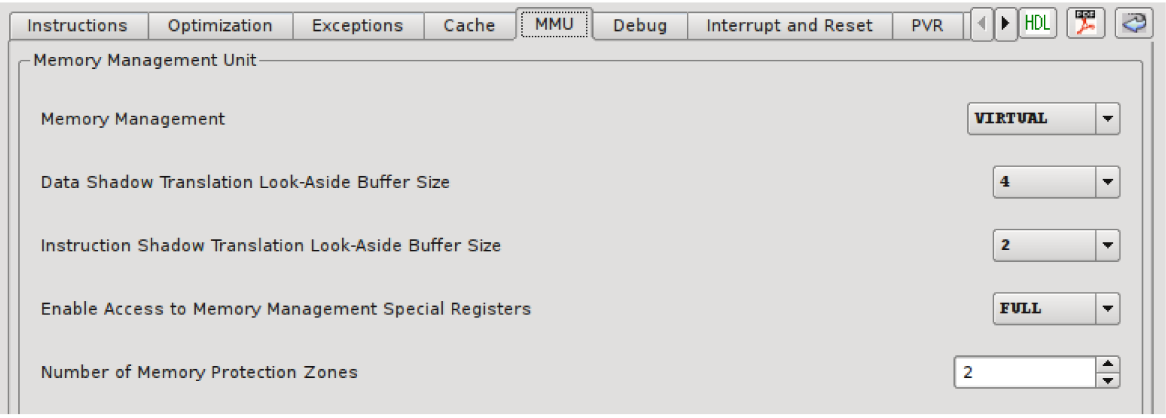
4.2 Konfigurace Hardware pro podporu OS Linux

Hardware vygenerovaný BSB průvodcem v EDK je nutné upravit, neboť OS Linux k běhu vyžaduje jisté změny. Nejdůležitější je úprava konfigurace mikroprocesoru Microblaze tak, aby podporoval běh operačního systému Linux. Jedná se především o povolení MMU se dvěma zónami ochrany paměti a povolení zpracování výjimek. To lze provést v EDK v záložce System Assembly View, zde vybereme IP komponentu microblaze_0 a dáme zobrazit kontextové menu, kde vybereme volbu Configure IP.

V dialogovém okně lze vlevo vybrat z předem připravených možností konfigurace Mikorprocesoru nebo tlačítkem Advanced zobrazit detailně všechny možné volby. V tomto případě je třeba nastavit následující.

- V záložce Exceptions – zapnout volby Data Side PLB, Instruction Side PLB, Illegal instructions, Unaligned Data, Generate Illegal Instruction Exception for NULL Instruction
- V záložce MMU – Memory Management=Virtual, Number of Memory Protection Zones = 2 (obr. 16)

Dále je vhodné obdobným způsobem nastavit IP komponentu RS232_DCE tak, aby používala rychlost 115200bps. Tím dojde k výraznému zrychlení výpisů na sériový terminál. Dále ještě zkontrolujeme zda všechny komponenty používají přerušovací systém. A časovač musí být nastaven na dvojitý dvaatřicetibitový.



Obr. 16: Příklad konfigurace Microblaze

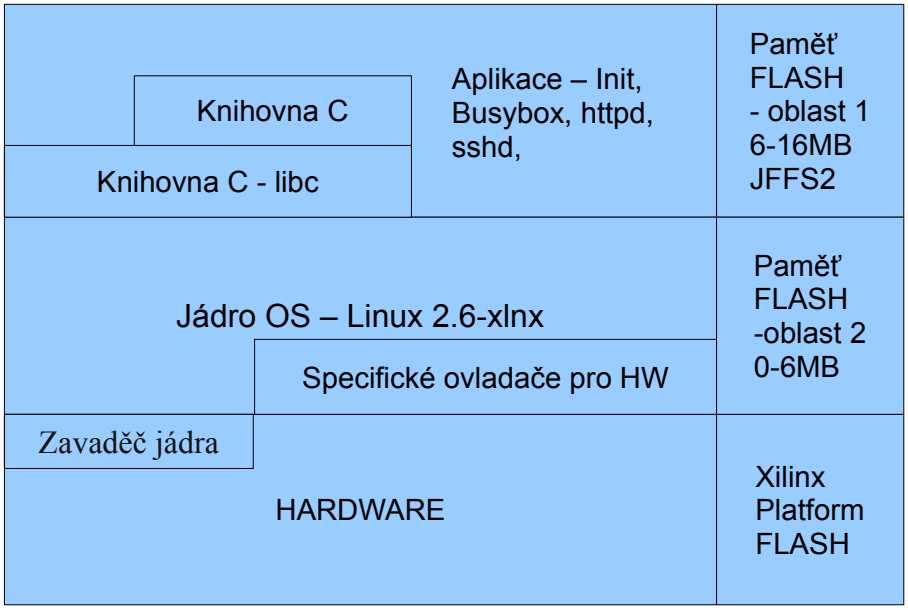
4.3 Paměťový model systému

Vhledem k tomu že systém obsahuje více druhů pamětí, je dobré znát paměťový model systému, neboť na něj navazují další části práce.

Systém je složen těchto pamětí:

- Xilinx Platform Flash
- BRAM
- DDR RAM
- FLASH

na vývojové desce je obsažena ještě paměť SPI Flash, ta ale v tomto návrhu není využita. Rozdělení pamětí z hlediska umístění částí návrhu je na obrázku 17.

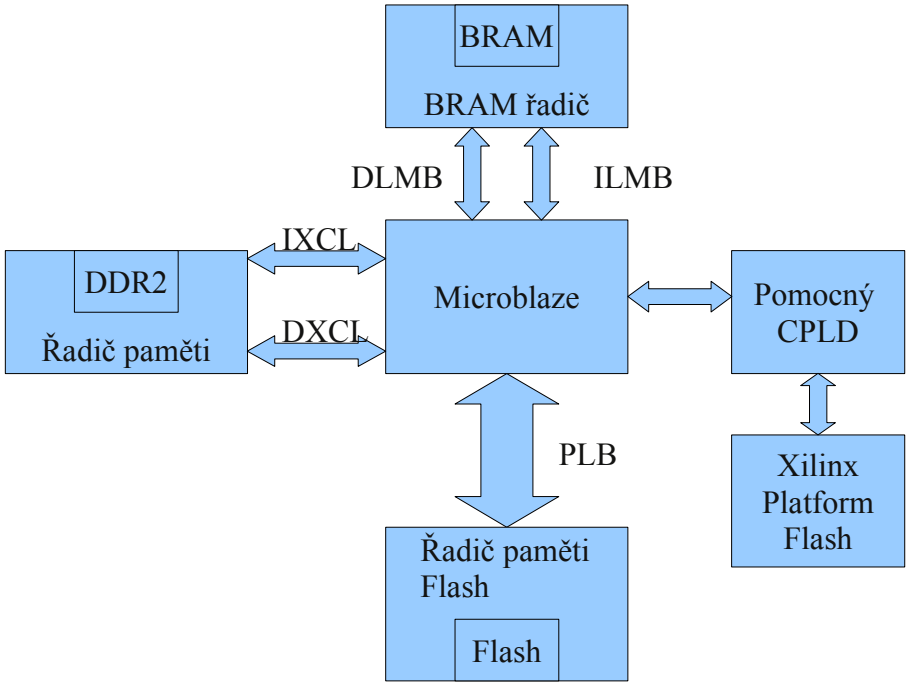


Obr. 17: Rozmístění návrhu v pamětech

Paměť Xilinx Platform Flash slouží k uložení návrhu zapojení hardware, obsah této paměti se po zapnutí napájení kopíruje do FPGA obvodu. Součástí hardwarového návrhu je i počáteční obsah BRAM, v tomto projektu je to zavaděč jádra operačního systému, a tedy i tento je umístěn v paměti Platform Flash.

Dále návrh používá paměť BRAM, která je umístěna přímo v obvodu FPGA, používají ji některé IP komponenty, zejména mikroprocesor Microblaze. Ten ji využívá jako lokální paměť a jako instrukční i datové vyrovnávací paměti cache. Paměť DDR je využita jako operační paměť. Zde jsou umístěny běžící aplikace a jejich data. V paměti FLASH je uloženo jádro systému a

kořenový systém souborů, ten obsahuje aplikace, konfigurační soubory, odkazy zařízení, obsah Webových stránek, knihovny a vše potřebné k běhu. Tato paměť je rozdělena na dvě části pomocí MTD. V první části je umístěno jádro operačního systému, tato část paměti není formátována žádným souborovým systémem. Druhá část pak využívá souborového systému JFFS2, který je speciálně určen k práci na zařízení typu MTD. Systém MTD i JFFS2 bude vysvětleno dále.



Obr. 18: Připojení pamětí k mikroprocesoru

Obrázek 18. pak popisuje propojení pamětí s mikroprocesorem Microblaze. Paměť BRAM je připojena pomocí dvouportového řadiče přes sběrnici LMB, ta je rozdělená na instrukční ILMB a datovou DLMB. Paměť DDR RAM je připojena pomocí také dvouportového řadiče přes sběrnici XLC, ty jsou zvlášť pro datovou DXCL a instrukční IXCL část. Dále je připojena paměť FLASH, ta je připojena sběrnicí PLB, která slouží i ostatním komponentám systému. Paměti nejsou připojeny přímo, ale vždy přes řadič. Tento řadič je IP komponentou a tedy implementován v FPGA obvodu.

Paměť Xilinx Platform Flash je pak připojena pomocí pomocného prvku CPLD přímo k FPGA obvodu (na obrázku orientačně připojeno přímo k mikroprocesoru). Tento CPLD slouží k výběru a propojení různých pamětí, na vývojové desce lze totiž FPGA obvod nahrávat z pamětí Flash, SPI Flash, Xilinx Platform Flash či přes JTAG rozhraní. Obsah tohoto CPLD prvku je dodáván s vývojovou deskou a pro jeho správnou funkci je nutné v něm mít nahrán tento návrh.

5. Návrh Software

Návrh software tohoto systému se skládá z několika části. První částí je zprovoznění operačního systému Linux, jež tvoří základ celého systému. Dále je popsán princip zavadění jádra a vytvoření zavaděče operačního systému, který je nezbytný k tomu, aby celkový návrh na vývojové desce byl schopen samostatného startu. Další část se týká zprovoznění aplikace Busybox, která je minimalizovanou a zjednodušenou implementací známých linuxových aplikací (jako je init, iproute2, cat, echo, telnet, vi, atd.). Tuto aplikaci využívají téměř všechny vestavěné systémy, které využívají operačního systému Linux. Součástí aplikace je také služba httpd, kterou využiji ke zprovoznění webového serveru. Předposlední část se věnuje vytvoření kořenového souborového systému a základní konfigurací linuxového systému a služeb. Nakonec je kapitola věnující se umístění softwarového návrhu do pamětí. Detailní informace o vytváření systému lze nalézt v literatuře [6], obecnější pak v [11].

5.1 Zprovoznění operačního systému Linux

Cílem zprovoznění systému Linux je vytvoření spustitelného obrazu jádra a zkouška jeho běhu na vývojové desce. Jádro je potřeba správně nakonfigurovat a zahrnout do něj součásti, které bude návrh používat. To je například zapnutí podpory IP protokolu, sériového portu, IP komunikace, systému MTD či podpora souborového systému JFFS2. Dále je potřeba jádru předat informace o hardware na kterém má běžet, neboť architektura Microblaze nepodporuje funkce samostatného zjišťování používaného hardware (Plug and Play). Tyto informace poskytuje device-tree soubor. Jádro je vytvářeno metodou křížové kompilace, kterou nám umožní toolchain dodávaný firmou Xilinx.

Ke zprovoznění jsou potřeba následující kroky:

- device-tree soubor
- funkční GNU toolchain
- konfigurace a kompilace jádra
- Test - nahrání do RAM paměti, spuštění.

Jednotlivým částem se budu věnovat v následujících kapitolách.

5.1.1 Device-tree

Device-tree je textovým souborem, který popisuje konfiguraci hardware, a především jeho mapování do paměťového prostoru. K vygenerování device-tree souboru slouží tzv. Board Support Package, ten lze stáhnout ze stránek fy. Xilinx. Vzhledem k tomu, že tento soubor je popisem konfigurace Hardware, tak je nutné po každé jeho úpravě vygenerovat i nový device-tree soubor.

Postup pro vygenerování:

1. Stažení pomocí nástroje GIT:
   v základním adresáři projektu spustíme příkaz:
   `git clone git://git.xilinx.com/device-tree.git`

2. Otevřeme projekt v EDK, a zvolíme Project-Export Hardware Design to SDK, v otevřeném okně pak zvolíme Export&Lauch SDK

3. V otevřeném okně SDK zvolíme Xilinx Tools, Repositories, v okně pak v části local repositories zvolíme New a vybereme hlavní adresář projektu který obsahuje adresář bsp, poté je nutno SDK zavřít a znova otevřít

4. Nyní vybereme v Menu File New-Xilinx Board Support Package, z menu pak vybereme device-tree a potvrdíme Finish

5. Následně je třeba nastavit správně tento balíček, a to parametry bootargs na console=ttyUL0 root=/dev/ram, což pak linuxovému jádru dává informaci o tom kde má vypisovat hlášení na konzoli, v tomto případě ttyUL0 značí první adaptér typu UARTLite

6. Další volbou je console device, zde je nutno vyplnit Instanci IP zařízení, kde se budou vypisovat hlášení při startu systému, v našem případě je to RS232_DCE

7. Formulář potvrdíme OK, následně v levé části vybereme náš device_tree_bsp projekt, a vybereme v menu Project-Build All, tím se sestaví device-tree soubor.

8. Tento soubor pak nalezneme v adresáři projektu
   `SDK/SDK_Workspace_35/device-tree_bsp_0/microblaze_0/libsrc/device-tree_v0_00_x/xilinx.dts`

5.1.2 Kompilace a konfigurace jádra

Pro tento projekt bylo použito jádro 2.6.37-xlnx, které dodává firma Xilinx. Oproti čistému jádru z www.kernel.org obsahuje doplňující opravy a ovladače pro architekturu Microblaze.

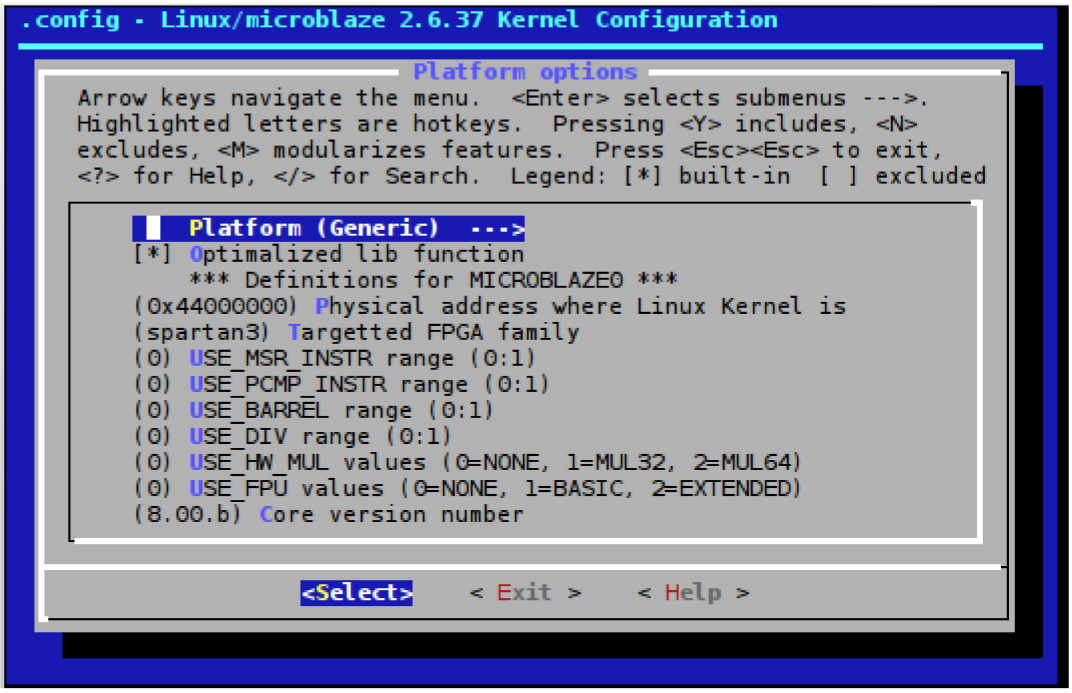
Jádro stáhneme zadáním tohoto příkazu v hlavním adresáři projektu:

```
git clone git://git.xilinx.com/linux-2.6-xlnx.git
```

Konfigurace: konfigurace probíhá podobně jako u např. architektury x86 a to pomocí příkazu:

```
make ARCH=microblaze menuconfig
```

Následující obrázek 19. je příkladem zobrazení konfiguračních voleb, v tomto případě se jedná o nastavení použitého mikroprocesoru.



Obr. 19: Příklad konfigurace jádra

Tímto postupem lze projít všechny položky a nastavit jádro. Je však ale vhodnější použít před-konfiguraci, kterou firma Xilinx poskytuje se zdrojovými kódy jádra, toho docílíme příkazem:

```
make ARCH=microblaze xilinx_mmu_defconfig.
```

Jméno těchto konfiguračních souborů získáme nahlédnutím do adresáře arch/microblaze/configs zdrojových kódů jádra. Následně pak můžeme spustit konfiguraci s volbou menuconfig a upravit konfiguraci. Celá konfigurace je poměrně obsáhlá (cca.1000 řádků), proto zde nemůže být uveden celý konfigurační soubor. Mnou použitý konfigurační soubor je tedy součástí přílohy na CD.

Následně je třeba nakopírovat vygenerovaný device-tree soubor do adresáře `arch/microblaze/boot/dts` zdrojových kódů jádra.

Kompilaci jádra pak spustíme příkazem:

```
make ARCH=microblaze simpleImage.xilinx
```

kde xilinx je odpovídá device-tree souboru xilinx.dts. Výstupem je pak ELF soubor simpleImage.xilinx v adresáři arch/microblaze/boot. Tento soubor můžeme nyní vyzkoušet spustit pomocí ladícího programu XMD.

5.1.3 Test běhu jádra

spustíme XMD, zadáme `connect mb mdm`, tento příkaz zajistí připojení k debugovacímu rozhraní mikroprocesoru, zobrazí se následující výpis s informace o mikroprocesoru:

```
JTAG chain configuration
--------------------------------------------------
Device   ID Code         IR Length    Part Name
 1       01c22093            6         XC3S500E
 2       05046093            8         XCF04S
 3       06e5e093            8         XC2C64A

MicroBlaze Processor Configuration :
-------------------------------------
Version.............................8.00.b
Optimization........................Performance
Interconnect........................PLB_v46
MMU Type............................Full_MMU
No of PC Breakpoints................1
No of Read Addr/Data Watchpoints...0
No of Write Addr/Data Watchpoints..0
Instruction Cache Support..........on
Instruction Cache Base Address.....0x44000000
Instruction Cache High Address.....0x47ffffff
Data Cache Support.................on
Data Cache Base Address............0x44000000
```

```
Data Cache High Address.....0x47ffffff
Exceptions  Support.....on
FPU  Support.....off
Hard Divider Support.....off
Hard Multiplier Support.....on - (Mul32)
Barrel Shifter Support.....on
MSR clr/set Instruction Support....on
Compare Instruction Support.....on
Data Cache Write-back Support.....off

Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
```

Nyní lze do RAM paměti nahrát program a následně spustit, nahrání provedeme příkazem: `dow simpleImage.xilinx`, dále pak provedeme spuštění příkazem: `run`

Zobrazí se informace o ELF souboru, můžeme si povšimnout, zda je správně jádro kompilováno s MMU, neboť vidíme adresy již virtuální mimo fyzický rozsah, který je uveden na výpisu konfigurace výše.

```
XMD% dow simpleImage.xilinx
Downloading Program -- simpleImage.xilinx
        section, .text: 0xc0000000-0xc0278ce7
        section, .init.text: 0xc02fa000-0xc031a3cf
        section, .init.ivt: 0xc031d870-0xc031d897
        section, __fdt_blob: 0xc0278ce8-0xc027cce7
        section, .rodata: 0xc027d000-0xc02d2d1f
        section, .init.ramfs.info: 0xc02d2d20-0xc02d2d23
        section, __ksymtab: 0xc02d2d28-0xc02d725f
        section, __ksymtab_gpl: 0xc02d7260-0xc02d8def
        section, __ksymtab_strings: 0xc02d8df0-0xc02e5f33
        section, __param: 0xc02e5f34-0xc02e6fff
        section, __ex_table: 0xc02e7000-0xc02e7cef
        section, .sdata2: 0xc02e7cf0-0xc02e7fff
        section, .data: 0xc02e8000-0xc02f893f
        section, .init.data: 0xc031a3d0-0xc031d86f
        section, .init.setup: 0xc031d898-0xc031db1f
        section, .initcall.init: 0xc031db20-0xc031dd4f
        section, .con_initcall.init: 0xc031dd50-0xc031dd57
        section, .init.ramfs: 0xc031e000-0xc070c867
        section, .bss: 0xc070d000-0xc0736393
Setting PC with Program Start Address 0x44000000
System Reset .... DONE

XMD% run
Processor started. Type "stop" to stop processor

RUNNING> XMD%
```

Nyní je mikroprocesor spuštěn a na sériové konzoli uvidíme výpisy nabíhajícího jádra:

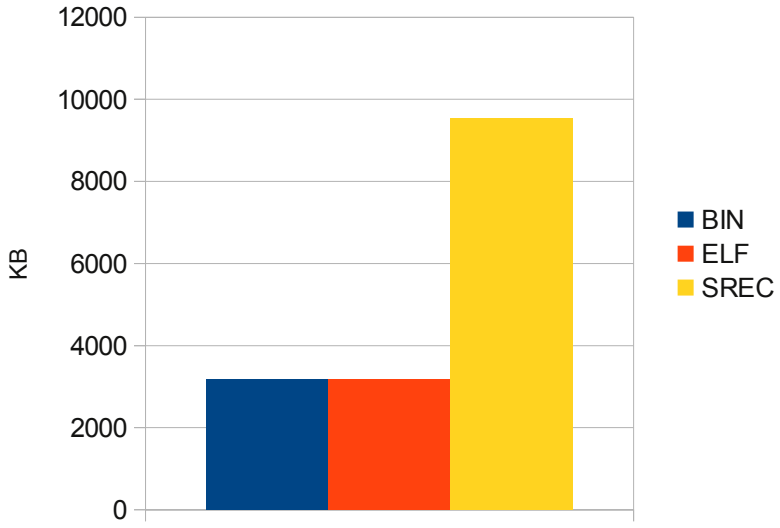
```
early_printk_console is enabled at 0x84000000
Ramdisk addr 0x00000003, Compiled-in FDT at 0xc0279cf0
!!!Your kernel not setup MSR instruction but CPU have it 0
ERROR: Microblaze BARREL, MSR, PCMP or DIV-different for kernel
and DTS
ERROR: Microblaze HW_MUL-different for kernel and DTS
Linux version 2.6.37-00715-gf5f5376 (frenk@localhost) (gcc version
4.1.2) #44 Sun Mar 27 22:13:58 CEST 2011
setup_cpuinfo: initialising
setup_cpuinfo: No PVR support. Using static CPU info from FDT
cache: wt_msr
setup_memory: max_mapnr: 0x4000
setup_memory: min_low_pfn: 0x44000
```

Zde si můžeme povšimnout chybového výpisu, který upozorňuje na nesoulad skutečné konfigurace mikroprocesoru a konfigurace uvedené při kompilaci jádra. V tomto případě však došlo k tomu, že mikroprocesor funkce obsahuje, ale jádro nikoliv. Tedy se připravujeme o možný vyšší výkon mikroprocesoru. V opačném případě by se mohlo stát, že jádro bude chtít využít funkce mikroprocesoru, které nejsou dostupné. Tento výpis byl pořízen účelově, v příloze na CD tak i finální realizaci, je již samozřejmě konfigurace správná.

5.3 Zavaděč (Bootloader)

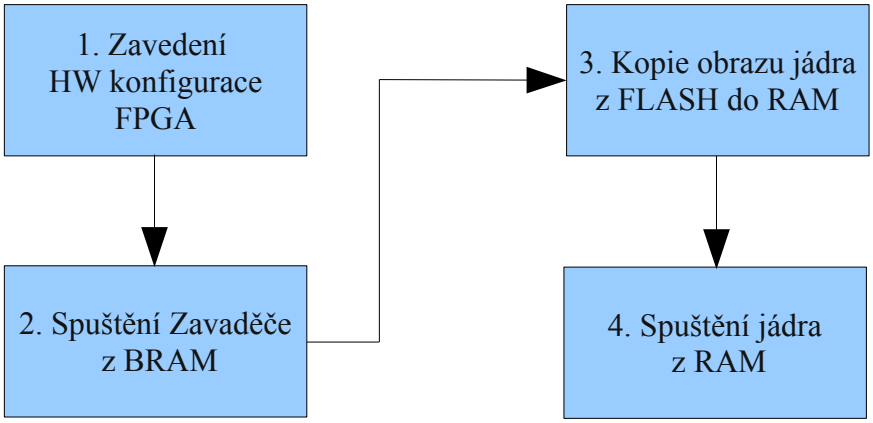
Postup výše uvedený sice vede ke spuštění jádra, nicméně toto je nutné provádět vždy po resetování obvodu. Proto jádro umístíme do paměti Flash a budeme jej z ní spouštět po resetování obvodu. K tomuto účelu slouží zavaděč, což je malý program, který načte obraz jádra do operační paměti a následně spustí jeho kód. Firma Xilinx takovýto zavaděč dodává v rámci SDK, nicméně se jedná o tzv. SREC zavaděč. Před jeho použitím je nutné obraz jádra převést do SREC formátu. Tento formát má však obrovskou nevýhodu a to obrovský nárust velikosti, například jádro o velikosti 3MB se po konverzi na SREC formát rozroste až na 10MB, tedy více než třikrát. Vzhledem k tomu že máme pouze 16MB paměti, je toto výrazné omezení. Nezbývalo by pak již místo pro umístění kořenového systému souborů, aplikací a webového obsahu. Toto lze obejít například použitím zavaděče, který umí načítat soubory ve formátu ELF, tyto zavaděče jsou však složité a velké. Nebylo by jej možné umístit do BRAM paměti. Výhodou takových zavaděčů jsou pokročilé funkce, které obsahují, a to například startování z serveru TFTP, FTP, BOOTP a podobně. Takovýmto zavaděčem často používaným u vestavěných systémů je třeba zavaděč U-BOOT. Avšak v tomto projektu je použito načítání přímo binárního obrazu jádra z vyhrazené části paměti FLASH, k tomuto účelu byl výraznou úpravou SREC zavaděče vytvořen vlastní jednoduchý zavaděč binárního obrazu jádra do operační DDR paměti.

Graf na obrázku 20. znázorňuje porovnání velikostí jednotlivých formátů. Zde vidíme, že použitím binárního formátu a zavaděče, dosáhneme oproti formátu SREC třetinové velikosti, a tím ušetříme i podstatnou část paměti Flash.



Obr. 20: Graf porovnání velikostí formátů obrazu jádra

Výhodou je jeho malá velikost a rychlost, je umístěn v BRAM paměti, která je součásti FPGA obvodu. Obsah této paměti je součástí souboru s návrhem zapojení FPGA obvodu, a je tedy umístěn v paměti Platform FLASH. Zavaděč je napsán v jazyce C, a v tomto případě se jedná o samostatný (standalone) softwarový návrh. Informace a dokumentaci k návrhům tohoto typu najdeme v literatuře [4]. Tento návrh byl vytvořen pomocí Xilinx SDK. Na dalších řádcích bude popsán a znázorněn postup vytvoření zavaděče. Na obrázku 21. je znázorněn postup zavádění jádra.

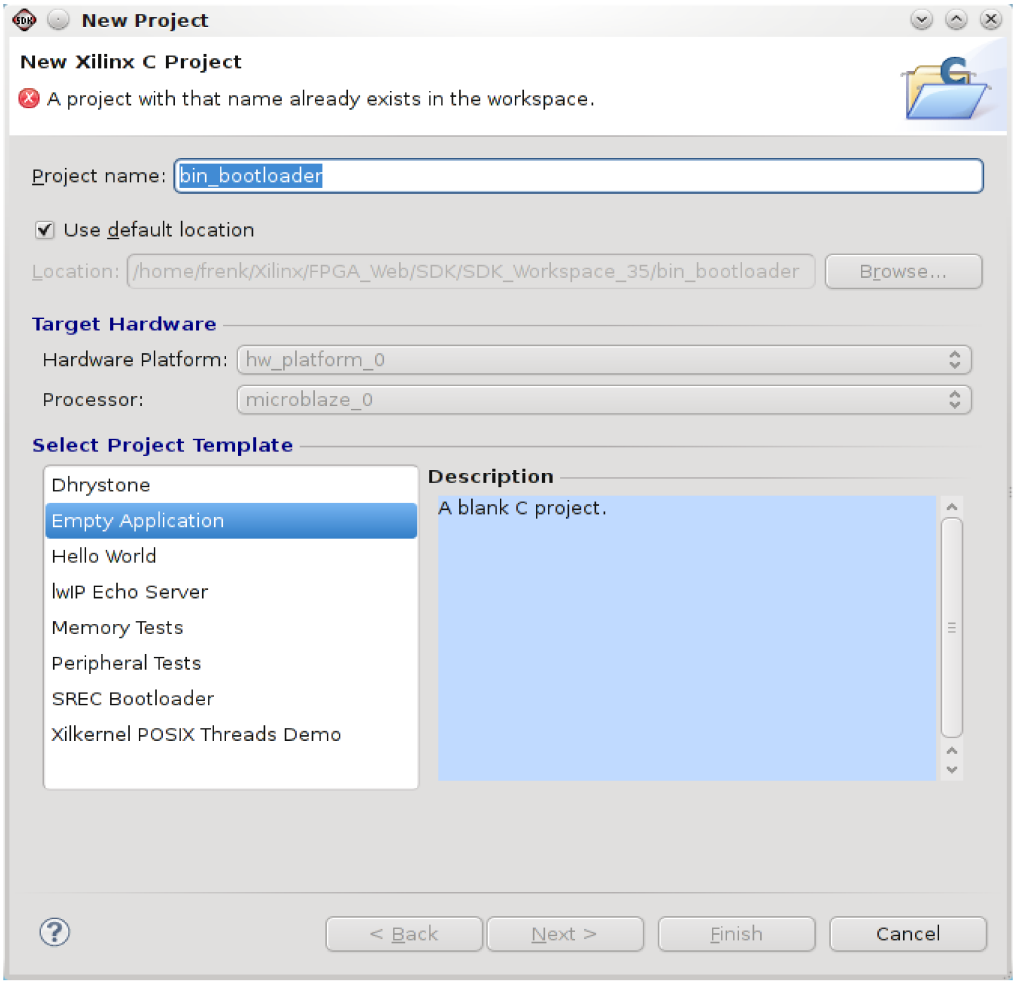


Obr. 21: Postup zavedení jádra

Postup vytvoření zavaděče:

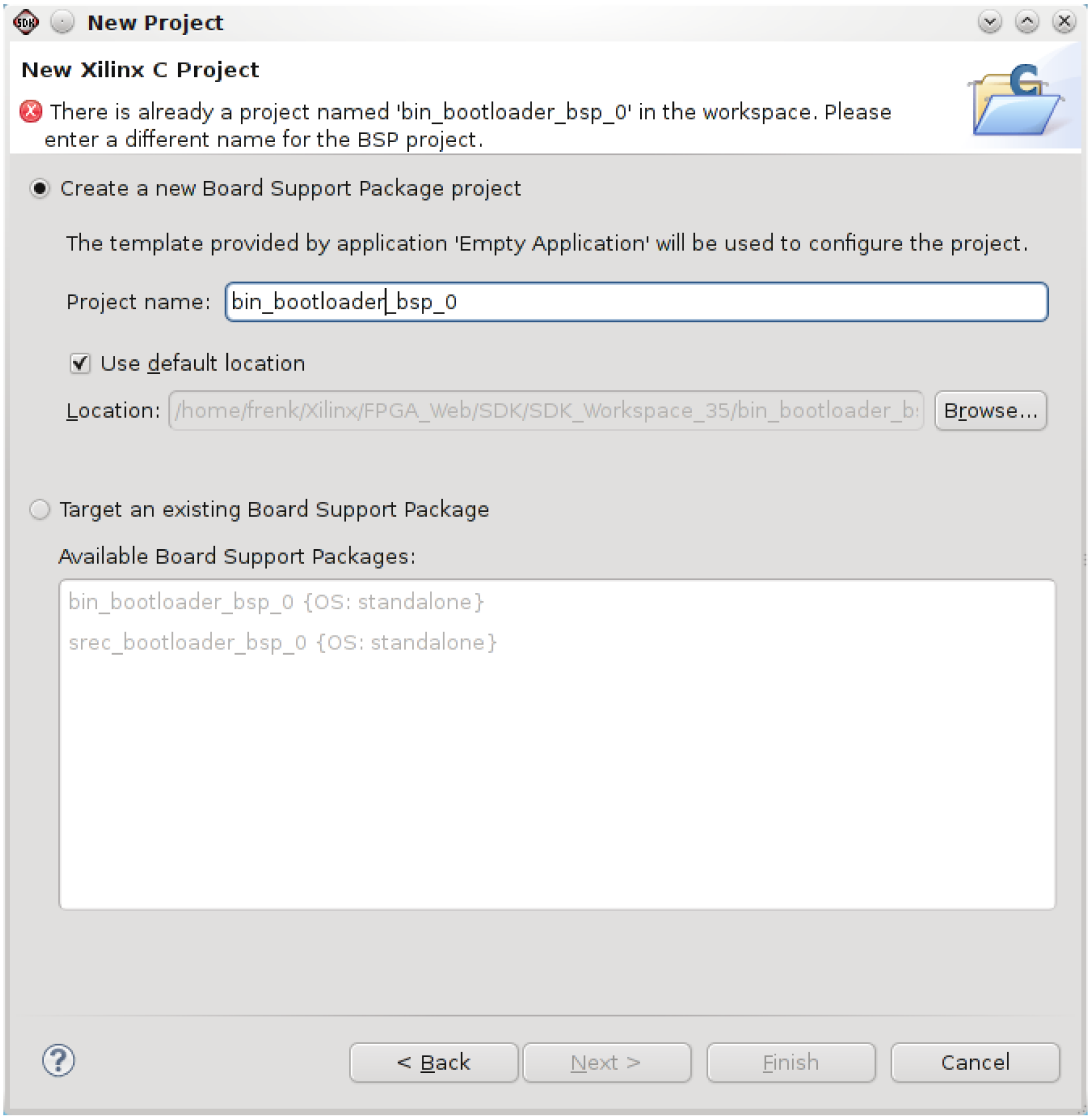
Začneme otevřením hardwarového návrhu v XPS, a zvolíme v menu Project položku Export Hardware Design to SDK. Otevře se okno, kde zvolíme Export & Lauch SDK. Po chvíli se vygenerují potřebné soubory a spustí prostředí SDK.

V prostředí SDK založíme nový projekt, vybereme v menu File – New – Xilinx C Project. Otevře se následující dialog jako na obrázku 22.



Obr. 22: Vytvoření nového SDK projektu

Zde můžeme v části Select Project Template vybrat některý z již vytvořených projektů. Zde vybereme Empty Application. Dále nahoře do pole název projektu bin_bootloader. Pokračujeme tlačítkem next na další dialog nastavení BSP (Obr. 23).



Obr. 23: Volba BSP

Zde vybereme Create New Board Support Package a zadáme název, ten by pro přehlednost měl odpovídat vytvářené aplikaci. Zadáme tedy bin_bootloader_bsp_0. Pokračuje stiskem tlačítka Finish, což vytvoří základ našeho projektu. Tímto postupem se BSP automaticky vytvoří jako Standalone. Stejným postupem také vytvoříme projekt SREC_bootloader, avšak zde zvolíme nikoliv Empty, ale použijeme stejnojmenný připravený Projekt. To proto, že využijeme některé jeho soubory, v projetu bin_bootloader.

Z projektu SREC_bootloader pak ze složky src překopírujeme do našeho projetu bin_bootloader taktéž do složky src následující soubory:

- blconfig.h
- bootloader.c
- platform_config.h
- platform.c
- platform.h
- portab.h

Z těchto souborů pak upravíme pouze bootloader.c, vše vymažeme a umístíme kód, který je součástí přílohy 2.

Dále je nutno vygenerovat Linker script, který obsahuje informace pro linker, především kde se má kód umístit. Tento skript vytvoříme v menu Xilinx – Generate Linker script, v následujícím dialogu pak vybereme ilmb_cntlr_dlmb_cntlr(což je řadič BRAM paměti) pro položky: Place Code Section, Place Data Section a Place Heap and Stack. Pokračujeme stiskem tlačítka Generate, což vytvoří soubor lscript.ld ve složce src. Nyní lze celý projekt zkompilovat třeba klávesovou zkratkou Ctrl+B.

5.4 Busybox

Je velmi oblíbená aplikace pro vestavěné systémy s operačními systémy Linux. Aplikace obsahuje všechny základní potřebné aplikace, které takové systémy mohou potřebovat. Tyto aplikace jsou vždy v zjednodušené implementaci a neobsahují například tolik funkcí či voleb, jako jejich plné implementace známé třeba z architektury x86. Velkou výhodou je velmi malá velikost těchto aplikací, ty jsou navíc tvořeny pouze jednou aplikací, a jednotlivé příkazy jsou pak v souborovém systému vytvořeny jako symbolické odkazy na tuto aplikaci. Výhodou také je, že máme jen jedninou aplikaci, kterou stačí zkompilovat a jen dodáním jádra, konfiguračních souborů získáme plně funkční Linuxový systém. Aplikaci lze navíc velmi snadné upravovat (konfigurovat), podobně jako linuxové jádro.

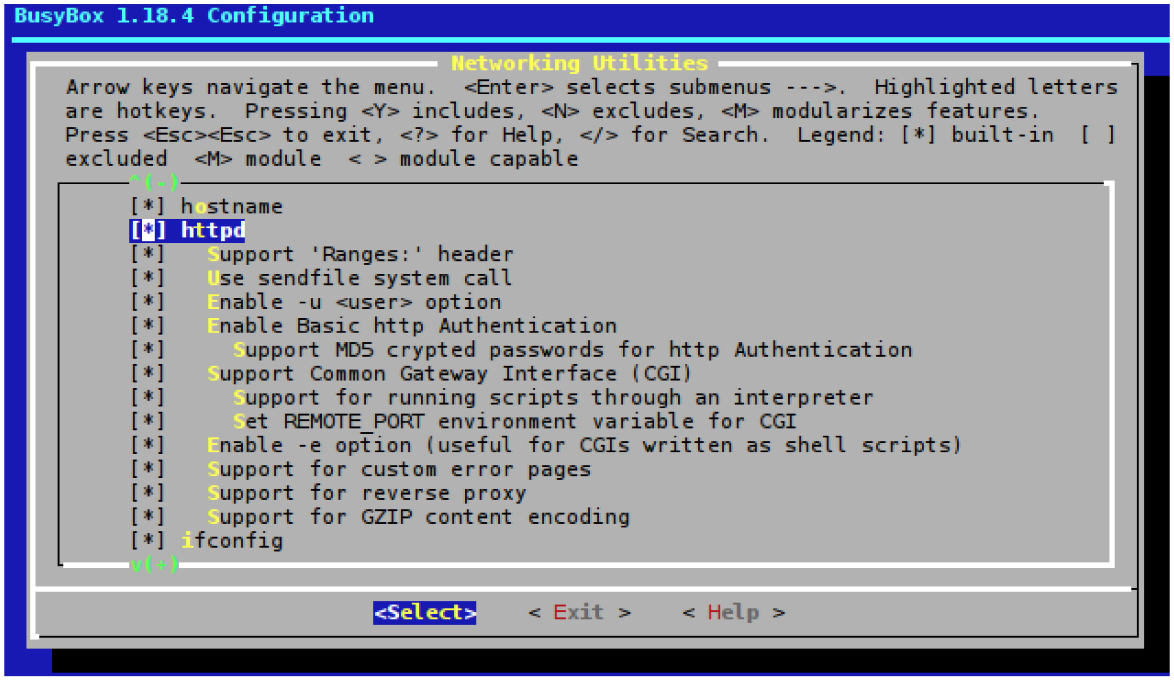
Aplikace Busybox je volně ke stažení a v mé realizaci byla použita verze BusyBox v1.18.4. Ke zprovoznění aplikace bude opět využito křížové kompilace, proto je nutné následující kroky provádět v shellu s nastaveným prostředím pro tyto kompilace.

Postup zprovoznění:

1. Stažení zdrojových kódů
   `wget http://www.busybox.net/downloads/busybox-1.18.4.tar.bz2`
2. Rozbalení `tar -xjvf busybox-1.18.5.tar.bz2`
3. Přejdeme do rozbaleného adresáře a spustíme konfiguraci:

`make ARCH=microblaze menuconfig`

4. Provedeme konfiguraci, především povolíme službu webového serveru httpd, telnetd pro vzdálený přístup, síťové aplikace dhcp, iproute2, podporu IPv4 aIPv6, konfiguračních voleb je opět velmi mnoho a výsledný soubor je k dispozici v CD příloze. Na obrázku 24. je příklad s konfigurací síťových vlastností.
5. Spustíme kompilaci `make ARCH=microblaze CC=mb-linux-gcc`
6. Provedeme instalaci `make install`



Obr. 24: Příklad konfigurace Busybox

Instalací se v adresáři s Busyboxem vytvoří adresář _install, ten obsahuje základ kořenového systému souborů operačního systému Linux. Ovšem chybí zde například některé konfigurační soubory, soubory odkazující na zařízení, či knihovny (například libc). Zprovozněním plně funkčního kořenového systému souborů se zabývá následující kapitola.

5.5 Vytvoření kořenového systému souborů

Základ kořenového systému se vytvoří již při instalaci Busyboxu, je však nutné kořenový systém souborů doplnit. Prvním krokem je přejmenování odkazu na program init (ten je také součásti Busyboxu), tento program je prvním, který se v Linuxovým systému spouští. Busybox správně vytvoří odkaz ovšem pod jiným jménem (linuxrc), než jádro očekává (init). Následujícími příkazy toto napravíme:

```
ln -s bin/busybox init

rm linuxrc
```

Dále potřebujeme vytvořit systémové adresáře a soubory:

```
mkdir sys
mkdir proc
mkdir lib
mkdir var
mkdir etc
mkdir tmp
mkdir mnt
mkdir root
mkdir home
mkdir dev
touch /etc/resolv.conf
```

Dále je třeba přidat potřebné knihovny, jako je především velmi potřebná libc, ty jsou součástí GNU_toolchain a stačí je nakopírovat:

```
cp -R  GNU_toolchain/mb_cross_compiler/microblaze-unknown-linux-
gnu/sys-root/lib/* /lib
```

Důležitou části je vytvoření konfiguračního souboru /etc/inittab, tento zpracovává proces init a jeho obsah je následující:

```
::sysinit:/etc/init.d/rcS
ttyUL0::respawn:/bin/sh
::restart:/sbin/init
```

První řádek zajistí spuštění skriptu rcS, jeho obsah bude uveden dále. Ten obsahuje skript, který spustí potřebné služby, připojí souborové systémy, nastaví síť. Druhý řádek potom spustí shell na sériové konzoli ttyUL0, což je RS232 port, který nám bude sloužit ke komunikaci s vývojovou deskou. Poslední řádek potom nastavuje restartování procesu init při jeho pádu či ukončení.

Obsah skriptu /etc/init.d/rcS:

```
#!/bin/sh

/bin/echo "Startuji rcS..."

/bin/echo "Vytvarimm body zarizeni "
/bin/mkdir /dev/pts
/bin/mount -t devpts devpts /dev/pts

/bin/echo "Pripojuji systemy souboru"
/bin/mount -t proc none /proc
/bin/mount -t sysfs none /sys
/bin/mount -t tmpfs none /tmp
```

```
/bin/echo "Spoustim dhcp klient"
/sbin/udhcpc -b

/bin/echo "Nastavuji hostname"
/bin/hostname -F /etc/hostname

/bin/echo "Startuji telnet server"
/usr/sbin/telnetd -l /bin/login
/bin/echo "Startuji web server"
/usr/sbin/httpd -h /var/www

/bin/echo "rcS Dokoncen"
```

Tento skript se postará o připojení nutných systému souborů, nastaví síť ze serveru DHCP, a nastartuje služby Telnet a Webový server.

Je potřeba také vytvořit potřebnou strukturu zařízení v /dev. Jedná se o odkazy na systémová zařízení, ty se skládají z minor a major čísla. Tyto čísla jsou uvedeny v dokumentaci jádra systému Linux, která je obsahem zdrojových kódů jádra. Následující příkazy je nutné provádět jako uživatel s právy root.

```
mknod console c 5 1
mknod mem c 1 1
mknod null c 1 3
mknod zero c 1 5
mkmod random c 1 8
mknod urandom c 1 9
mknod tty c 5 0
mknod tty0 c 4 0
mknod tty1 c 4 1
mknod tty2 c 4 2
mknod tty3 c 4 3
mknod tty4 c 4 4
mknod ttyUL0 c 204 187
mknod mtd0 b 31 0
mknod mtd1 b 31 1
```

Do kořenového systému byly také umístěny webové stránky. Jde o adresář /var/www a byly použity mé stránky se statickým obsahem věnující se mechanizmu RADIUS. Jdou dostupné na internetových stránkách http://homel.vsb.cz/~fra186/radius/ nebo přiloženém CD.

Tím máme vytvořen kořenový systém souborů, se kterým lze dále pracovat.

5.6 Umístění softwarového návrhu do paměti

Připraveny již máme všechny potřebné části a zbývá návrh umístit do paměti Flash, tak jak je uvedeno v kapitole věnující se paměťovému modelu. Tato paměť je v Linuxovém systému použita jako MTD. MTD je speciální typ zařízení, kdy máme přímý přístup k Flash paměti a nelze se ní chovat jako k blokovému či znakovému zařízení. Je především nutné respektovat velikost erase (mazacího) bloku, u těchto typů pamětí totiž nelze přímo přepisovat jakékoliv místo, je nutné načíst celý blok, upravit jej, místo na MTD zařízeni vymazat a následně na stejné místo uložit upravený blok. Zařízení typu MTD nepoužívá žádnou metodu, která by rovnoměrně využívala paměťové bloky. Paměti Flash mají pouze omezený počet přepisovacích cyklů a je žádoucí bloky co nejrovnoměrněji využívat, aby nedošlo častým přepisem ke zničení pouze části bloků a přitom zbytek by třeba nebyl nikdy využit. Více informací o MTD nalezneme v literatuře [9].

Ke správné funkci MTD je nutná správná konfigurace jádra a přidání aplikací pro použití MTD v aplikaci Busybox, obojí je již obsaženo v konfiguračních souborech v příloze na CD. Vhledem k tomu, že využíváme dvě oblasti paměti, je nutné provést jisté úpravy. Na zařízení není žádná tabulka oddílů a rozdělení paměti se zapisuje do souboru device-tree. V mém případě se jedná o oblast pro jádro 0-6MB a kořenový systém 6-16MB. Zde je vždy nutno dodržet správné zarovnání na erase bloky, v tomto případě se jedná o bloky velikosti 128KB (dáno použitým typem paměti Flash). To odpovídá velikosti 0x20000, a je nutné, aby paměťové adresy rozdělení byly dělitelné tímto číslem (zarovnány na bloky). Pro oblast s jádrem tedy budou použity adresy 0x0 až 0x60000 tedy velikost je 0x600000, oblast s kořenovým systémem pak bude 0x600000 až 0x1000000, velikost 0xA00000.

Část týkající se této paměti v device-tree souboru upravíme na konci definice Flash paměti následovně:

```
…
xlnx,xcl2-linesize = <0x4>;
xlnx,xcl2-writexfer = <0x1>;
xlnx,xcl3-linesize = <0x4>;
xlnx,xcl3-writexfer = <0x1>;
kernel@00000000 {
      label = "kernel";
      reg = <0x0 0x00600000>;
                };
rootfs@00600000 {
      label = "rootfs";
      reg = <0x00600000 0x01000000>;
      };
```

Tato úprava říká jádru, jak má paměť rozdělit. Po úpravě device-tree souboru je samozřejmě nutné znova spustit kompilaci jádra.

Kromě rozdělení paměti je třeba upravit část device-tree souboru, která obsahuje informace o spuštění pro jádro operačního systému. Tato část se nachází na počátku souboru a upravíme ji takto:

```
chosen {
           bootargs = "console=ttyUL0 loglevel=7
root=/dev/mtdblock1 rootfstype=jffs2 rw";
           linux,stdout-path = "/plb@0/serial@84000000";
```

Parametr bootargs nastavuje konzoli pro výpisy jádra na ttyUL0 což je IP komponenta sériového portu (UL = UART Lite), parametr root pak definuje, kde je umístěn kořenový souborový systém. V tomto případě je to tedy druhá oblast paměti Flash (číslovaní je od nuly). Také zde vidíme specifikaci použitého systému souborů JFFS2. Systém bude připojen v režimu pro zápis, což značí příznak rw.

První část paměti nepoužívá souborový systém, je možné do něj přímo zapsat binární obraz jádra. Ten je nejdříve nutné vytvořit, neboť při kompilaci se vytváří obraz typu ELF. K převodu formátu slouží následující příkaz:

```
mb-linux-objcopy -O binary simpleImage simpleImage.bin
```

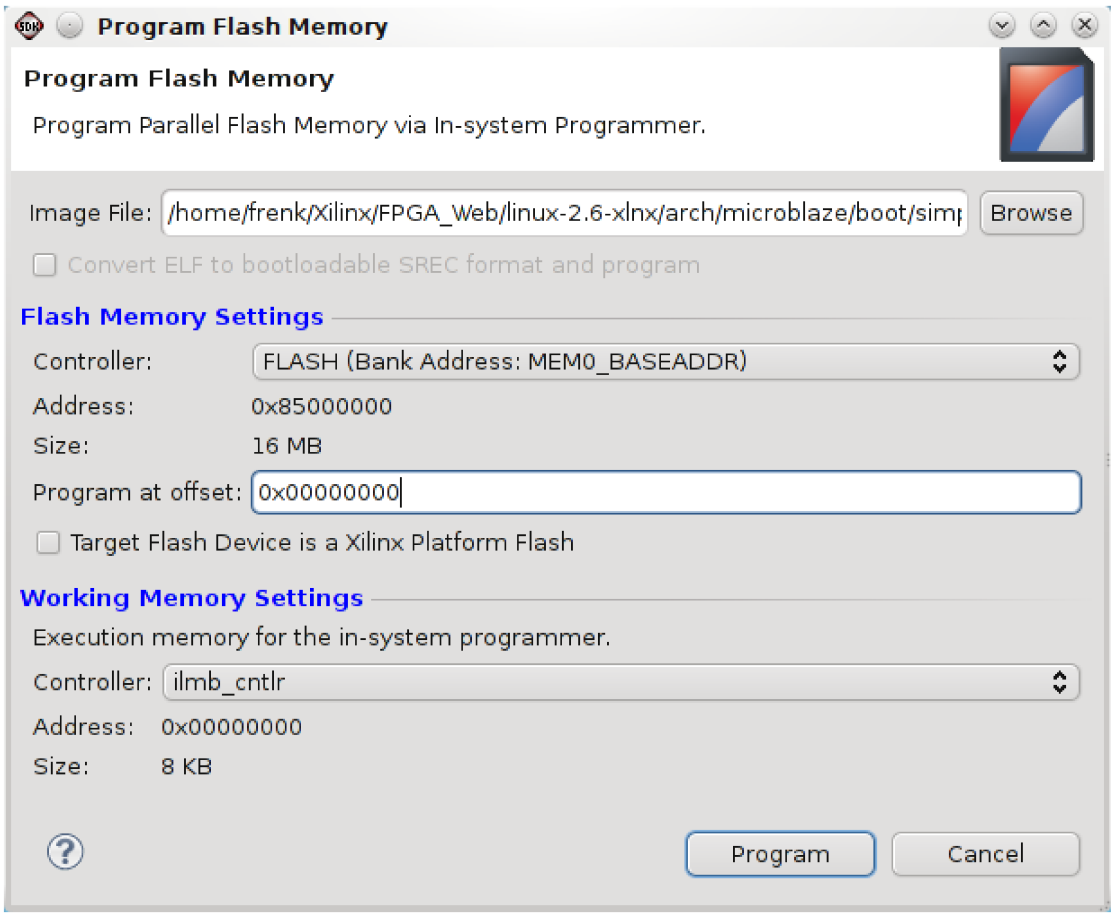
Vytvořený soubor simpleImage.bin je přímo připraven k zapsání do paměti, neboť zde nemáme souborový systém.

Druhou část paměti s obsahem kořenového souborového systému, ale přímo zapsat nelze. Je nutné použít souborový systém, takovým systémem pro MTD zařízení je JFFS2, který zajišťuje těmto zařízením techniky wear-leveling pro rovnoměrné využívání zápisu do bloků. Mimo to také podporuje kompresi dat. Pro jeho použití je opět nutná podpora v jádře operačního systému. Obraz souborového systému pak vytváříme na počítači příkazem:

```
mkfs.jffs2 -r busybox-1.18.4/_install/ -b -e 128KiB -o rootfs.img
```

použitím volby -r specifikujeme kořen našeho systému souborů, ze kterého budeme vytvářet obraz. Volba -b určuje endianitu (pořadí významnosti bajtů) cílového systému, v případě Microblaze je to Big-Endian. Volba -e specifikuje velikost erase bloku na 128KB. Poslední volba -o pak zajistí přesměrování výstupu do souboru rootfs.img. Tento soubor pak budeme nahrávat do druhé oblasti paměti Flash.

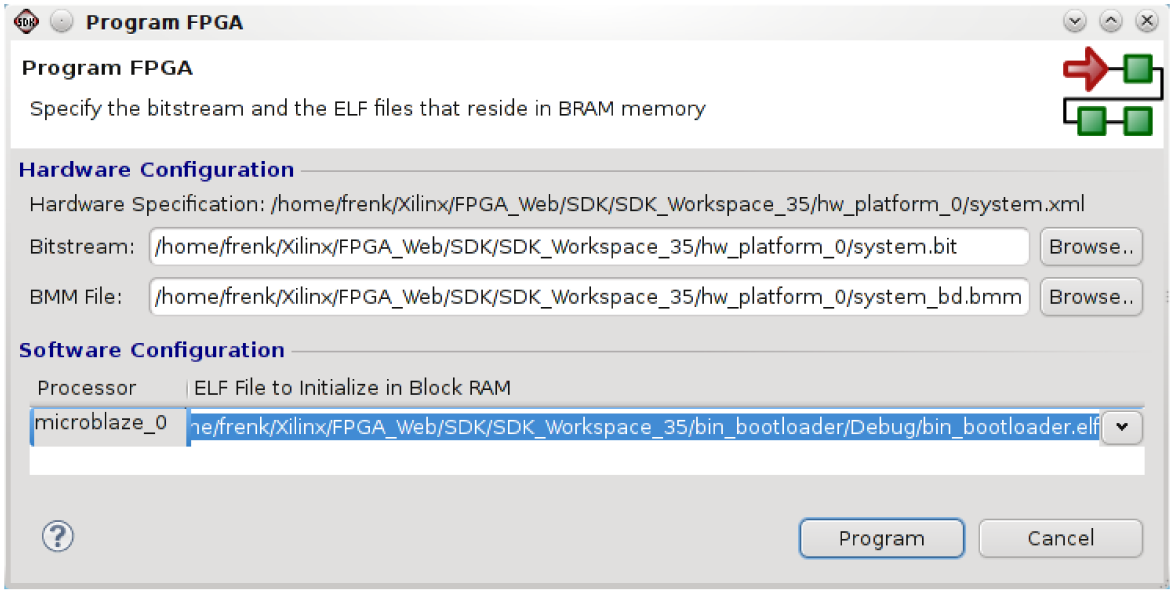
Nahrávání obsahu do paměti provádíme v prostředí SDK. Zde je k dispozici v menu Xilinx Tools nástroj Program Flash. Po jeho spuštění získáme dialog jako na obrázku 25.



Obr. 25: Dialog pro zápis do Flash paměti

Nejdříve umístíme obraz jádra, stiskem Browse otevřeme souborový dialog a vybereme připravený soubor simpleImage.bin. Jinak nic neměníme, tím se zapíše jádro na počátek paměti Flash. Zápis zahájíme stiskem tlačítka Program. Obdobně zapíšeme i obraz kořenového systému souborů, vybereme soubor rootfs.img a do pole Program at Offset vyplníme 0x00600000, což odpovídá počátku druhé oblasti paměti. Vytváření obrazů a popis programovaní je dobře popsán v literatuře [7]

Nyní je téměř celý softwarový návrh umístěn v paměti Flash a chybí již pouze uložit do paměti Platform Flash konfiguraci hardware (bitstream soubor). Součástí je i zavaděč operačního systému. Nejdříve je nutné opět v SDK vytvořit bitstream, jehož součástí bude i zavaděč. V SDK zvolíme menu Xilinx Tools – Program FPGA. V otevřeném dialogu pak v části „ELF File to Initialize in Block BRAM“ zvolíme náš vytvořený bin_bootloader. Dialog je znázorněn na obrázku 26.



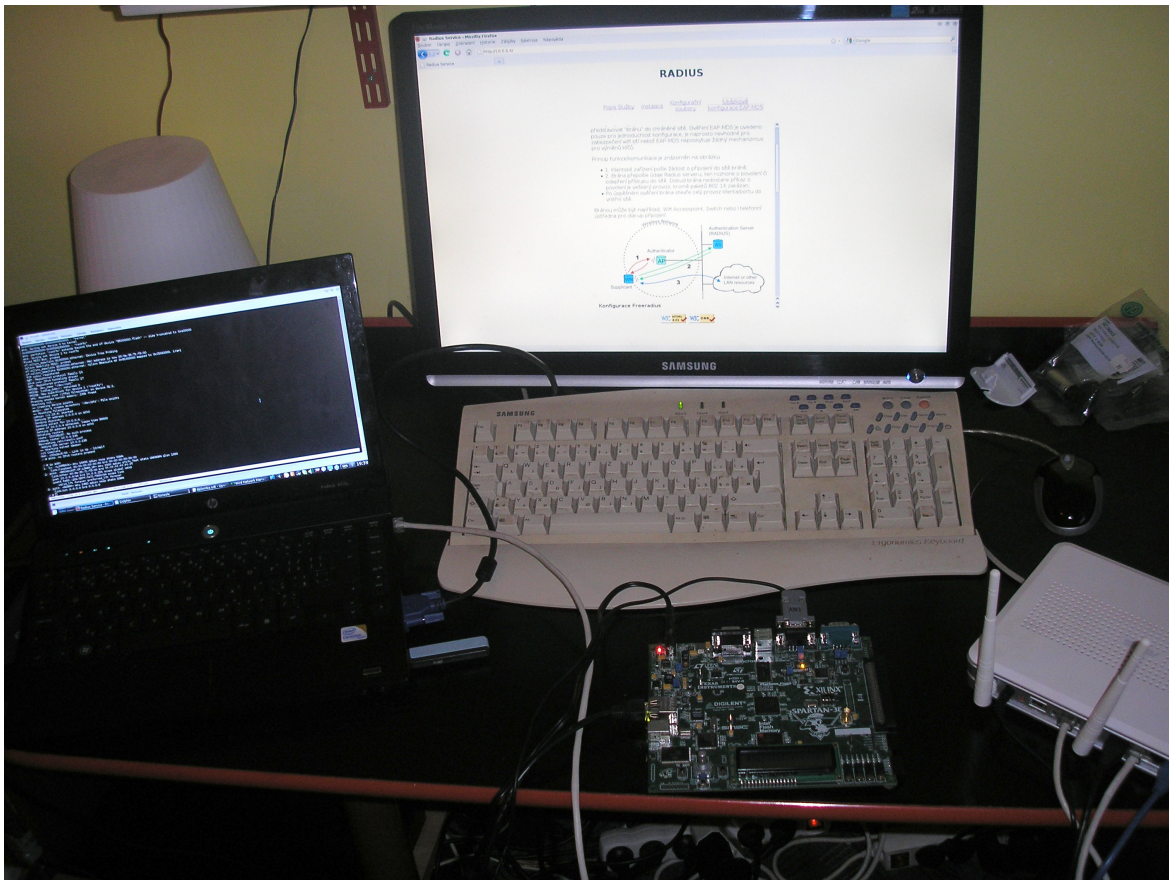
Obr. 26: Dialog pro vytvoření HW návrhu s obsahem BRAM

Výsledný soubor s konfigurací hardware i zavaděčem se vytvoří po stisku tlačítka Program. Pokud nemáme k počítači připojenou vývojovou desku, obdržíme hlášení, že se programovaní nezdařilo. To však nevadí, neboť se soubor i tak vytvoří. Pokud je vývojová deska připojena FPGA obvod se nakonfiguruje a spustí. Výsledný soubor nalezneme jej v adresáři `SDK/SDK_Workspace_35/hw_platform_0/download.bit`.

Naprosto posledním krokem je naprogramování Xilinx Platform Flash výše uvedeným souborem s návrhem zapojení i zavaděčem. Tak docílíme toho, že vývojová deska bude sama i po výpadku napájení fungovat, protože při zapnutí si překopíruje obsah této paměti do obvodu FPGA. Tento postup je velmi dobře popsán v uživatelské příručce [12] programu Impact, který je také součástí EDK. Proto zde není přesně uveden.

6. Odzkoušení systému

Posledním bodem návrhu systému je samozřejmě jeho vyzkoušení. K tomu bylo využito notebooku, směrovače (slouží jako DHCP sever, a pro připojení k internetu) a samozřejmě vývojové desky. Fotografii zapojení z průběhu zkoušení lze vidět na obrázku 27. Na monitoru vlevo můžeme vidět běžící konzoli vývojové desky přes sériový port. Na monitoru vpravo lze vidět webové stránky zobrazené z vývojové desky. Zkoušení proběhlo následovně. Po naprogramování všech pamětí, je nutné správně přepnout dle manuálu [1] vývojovou desku tak, aby startovala z paměti Platform Flash. Poté byla deska připojena k sériovému portu počítače, připojen Ethernetový kabel k počítačové síti, která umí poskytovat adresy ze serveru DHCP. Po zapnutí vývojové desky začne systém nabíhat, o čemž se můžeme přesvědčit připojením na sériový port (program minicom, hyperterminál, putty), kde musíme vidět výpisy jádra a nakonec se systém přepne do systémové konzole.



Obr. 27: Fotografie zkoušení návrhu

V systémové konzoli se můžeme podívat na přidělenou IP adresu příkazem ip addr či ifconfig. Poté byla vyzkoušena příkazy ping a ping6 funkčnost síťového připojení. Následně byl vyzkoušen přístup k webovým stránkám pomocí webového prohlížeče v počítači připojeném na stejný segment sítě. Webové stránky se správně načítají a fungují jak pomocí IP protokolu verze 4, tak i pomocí protokolu verze 6. Tímto byla plně otestována požadovaná funkčnost implementace návrhu, který byl hlavním cílem této práce.

Byly také pořízeny následující výpisy z konzole běžícího systému vývojové desky. Jedná se o výpis verze operačního systému, informace o procesoru, IP adresy, informace o MTD, využití paměti a výpis běžících procesů. Zde si můžeme všimnout běžící služby httpd, která reprezentuje webový server.

Verze operačního systému:

```
/ # uname -a
Linux FPGA_WEB 2.6.37-00715-gf5f5376 #44 Sun Mar 27 22:13:58 CEST
2011 microblaze GNU/Linux
```

Informace o systému MTD:

```
/ # cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00600000 00020000 "kernel"
mtd1: 00a00000 00020000 "rootfs"
```

Využití kořenového systému souborů:

```
/ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/root                10240      6004      4236  59% /
```

IP adresy:

```
/ # ip addr
1: lo: <LOOPBACK> mtu 16436 qdisc noop state DOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UNKNOWN qlen 1000
    link/ether 00:0a:35:71:f9:00 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.4/24 brd 10.0.0.255 scope global eth0
    inet6 fe80::20a:35ff:fe71:f900/64 scope link
       valid_lft forever preferred_lft forever
3: sit0: <NOARP> mtu 1480 qdisc noop state DOWN
    link/sit 0.0.0.0 brd 0.0.0.0
```

Informace o mikroprocesoru:

```
/ # cat /proc/cpuinfo
CPU-Family:     MicroBlaze
FPGA-Arch:      spartan3
CPU-Ver:        Unknown, big endian
CPU-MHz:        50.00
BogoMips:       23.96
HW:
 Shift:         yes
 MSR:           yes
 PCMP:          yes
 DIV:           no
 MMU:           3
 MUL:           v1
 FPU:           no
 Exc:           unal ill
Icache:         2kB     line length:    16B
Dcache:         8kB     line length:    16B
                write-through
HW-Debug:       yes
PVR-USR1:       00
PVR-USR2:       00000000
Page size:      4096
```

Výpis běžících procesů:

```
/ # ps -e
PID   USER     TIME   COMMAND
    1 root       0:08 init
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
    5 root       0:00 [kworker/u:0]
    6 root       0:00 [khelper]
    7 root       0:00 [sync_supers]
    8 root       0:00 [bdi-default]
    9 root       0:00 [kblockd]
   10 root       0:02 [kswapd0]
   11 root       0:00 [fsnotify_mark]
   12 root       0:00 [aio]
   13 root       0:00 [crypto]
   17 root       0:00 [kworker/u:1]
   22 root       0:00 [mtdblock0]
   23 root       0:00 [mtdblock1]
   26 root       0:06 [jffs2_gcd_mtd1]
   29 root       0:03 [kworker/0:1]
   45 root       0:00 /sbin/udhcpc -b
   49 root       0:00 /usr/sbin/telnetd -l /bin/login
```

```
   51 root       0:00 /usr/sbin/httpd -h /var/www
   53 root       0:01 /bin/sh
   78 root       0:00 [kworker/0:2]
   79 root       0:00 [kworker/0:0]
   81 root       0:00 [flush-mtd-unmap]
   83 root       0:00 ps -e
```

Využití paměti:

```
/ # cat /proc/meminfo
MemTotal:          61636 kB
MemFree:           57260 kB
Buffers:               0 kB
Cached:             2176 kB
SwapCached:            0 kB
Active:              836 kB
Inactive:           1512 kB
Active(anon):        172 kB
Inactive(anon):        0 kB
Active(file):        664 kB
Inactive(file):     1512 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:           196 kB
Mapped:              532 kB
Shmem:                 0 kB
Slab:                  0 kB
SReclaimable:          0 kB
SUnreclaim:            0 kB
KernelStack:         200 kB
PageTables:            0 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:       30816 kB
Committed_AS:       1732 kB
VmallocTotal:     950272 kB
VmallocUsed:       16876 kB
VmallocChunk:     933180 kB
```

7. Závěr

V rámci této diplomové práce se podařilo vytvořit plně funkční vestavěný systém, který plní funkci Webového serveru. Je schopen komunikovat přes Ethernetové rozhraní, a to protokoly IPv4 i IPv6, což stále i v dnešní době nástupu IPv6 není u spousty profesionálních zařízení běžné. Celá realizace na vývojové desce je samostatně schopná jak naběhnutí, tak i běhu. Mimo zadání práce na lze k systému i vzdáleně přistupovat a to protokoly Telnet, SSH, či FTP. Celá práce velmi detailně mapuje postup vytvoření takového systému, takovýto kompletní dokument je chybějící části dostupné literatury. To velmi snadno umožní na tuto práci navázat a dále pokračovat ve vývoji podobných systémů s FPGA prvky. Celá implementace také využívá maximálně systému SoC, kdy celý návrh mimo paměti a čip fyzické vrstvy je umístěn pouze v jediném FPGA obvodu a to i přes to, že se jedná o jeden z nejmenších obvodů FPGA, které firma Xilinx vyrábí. Ale i v takovémto obvodu se podařilo vytvořit celý vlastní počítač.

Tento systém je samozřejmě ukázkový. Jeho snadným doplněním lze pokračovat ve vývoji například převodníků rozhraní Ethernet – RS-232, I2C, SPI, CAN. Také lze tento návrh využít společně s jiným hardwarovým návrhem v FPGA obvodu. Například OFDM modulátor či AAC enkodér, a část navrženou v této práci využít například pro jejich vzdálené řízení či předávání výstupů do počítačové sítě či Internetu. Zvláště uživatelsky přívětivé ovládání pomocí webových stránek je u těchto zařízení stále častěji používáno. Díky použití FPGA obvodů lze tento systém kdykoli doplnit nejen po softwarové, ale i hardwarové stránce. Můžeme například v návrhu nechat několik volných vstupů/výstupů. A využít je až během provozu zařízení, třeba pro přidání sběrnice I2C. K tomu bude stačit pouhá změna obsahu pamětí. To u jiných systémů není možné.

Takovéto systémy jsou dnes špičkovým návrhem požívaným v mnoha oblastech průmyslu, od řídících jednotek aut, počítačových komponent, lékařské přístroje až po řízení raket či letadlových systémů. Také si lze všimnout, že u těchto moderních návrhů se již nezajímáme o nízkoúrovňové části návrhu. Systém se skládá z jednodušších bloků (komponent), které jsou již otestovány a navenek plní svou funkci, přitom jejich vnitřní struktura nás nezajímá. U těchto návrhů se tedy můžeme soustředit přímo na tvorbu vlastního návrhu propojováním a konfigurováním již vytvořených komponent. To velmi usnadňuje a urychluje návrh, neboť se nemusíme zabývat již vymyšlenými a funkčním částmi a můžeme se spoutředit na implementaci vlastních nových návrhů. Nicméně pokud bychom potřebovali je zde stále možnost některé části vyvíjet i na nižších úrovních.

Systém byl kromě vedoucího práce předveden také studentům VŠB v rámci předmětu Programovatelné Logické Prvky, kde se studenti učí základní návrhy s hradlovými poli. To snad povede k vyššímu zájmu studentů o tento obor, který v současné době není na naší škole moc rozvíjen a snad i v budoucnu k navázání na tuto práci. Tato práce je koncipována tak, aby mohla sloužit i jako studijní podpora při tvorbě vestavěných systémů s FPGA obvody.

Seznam použité literatury

[1] Xilinx.com [online]. 2011 [cit. 2011-05-02]. Spartan-3E FPGA Starter Kit Board User Guide. Dostupné z WWW: <http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf>.

[2] Xilinx.com [online]. 2010 [cit. 2011-05-02]. Embedded System Tools Reference Manual. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/est_rm.pdf>.

[3] Xilinx.com [online]. 2010 [cit. 2011-05-02]. Installation, Licensing and Release Notes. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/irn.pdf>.

[4] Xilinx.com [online]. 2010 [cit. 2011-05-02]. OS and Libraries Document Collection. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/oslib_rm.pdf>

[5] Xilinx.com [online]. 2010 [cit. 2011-05-02]. MicroBlaze Processor Reference Guide. Dostupné z WWW: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/mb_ref_guide.pdf>.

[6] YAGHMOUR, Karin. Kerneltravel.com [online]. 2003 [cit. 2011-05-02]. Building Embedded Linux Systems. Dostupné z WWW: <http://www.kerneltravel.net/downloads/Building.Embedded.Linux.Systems.pdf>.

[7] Xilinx.com [online]. 2009 [cit. 2011-05-02]. Using and Creating Flash Files for the MicroBlaze Development Kit - Spartan- 3A DSP 1800A Starter Platform. Dostupné z WWW: <http://www.xilinx.com/support/documentation/application_notes/xapp1106.pdf>.

[8] Ubuntuforums.org [online]. 2010 [cit. 2011-05-02]. Xilinx ISE WebPack 12.2 on Ubuntu 10.04 LTS. Dostupné z WWW: <http://ubuntuforums.org/archive/index.php/t-1547435.html>.

[9] Infradead.org [online]. 2008 [cit. 2011-05-02]. Memory Technology Devices. Dostupné z WWW: <http://www.linux-mtd.infradead.org/doc/general.html>.

[10] Pandatron.cz [online]. 2008 [cit. 2011-05-02]. CPLD a FPGA 1.díl - popis obvodů. Dostupné z WWW: <http://pandatron.cz/?481&cpld_a_fpga_1.dil_-_predstaveni_obvodu>.

[11] Xilinx Open Source Wiki [online]. 2011 [cit. 2011-05-02]. Device Tree Generator. Dostupné z WWW: <http://xilinx.wikidot.com/device-tree-generator>.

[12] Xilinx.com [online]. 2010 [cit. 2011-05-02]. iMPACT User Guide. Dostupné z WWW: <http://www.xilinx.com/itp/xilinx4/pdf/docs/pac/pac.pdf >.

Obsah CD

Součástí této práce příloha na CD. Na něm je umístěna tato práce v PDF formátu, a jsou zde přiloženy důležité konfigurační soubory, obsah webových stránek, zdrojové kódy zavaděče, a obrazy pamětí s finálním návrhem.

Struktura:

```
Bitstream/
        download.bit            - Bitstream soubor s návrhem hardware

bootloader/
        src/
                *.c, *.h        -  zdrojové kódy zavaděče
Busybox/
        config                  - konfigurační soubor BusyBoxu

Linux/
        config                  - konfigurační soubor jádra
        simpleImage.bin         - zkompilovaný obraz jádra
        xilinx.dts              - device-tree soubor

RootFS/
        rootfs.img              - JFFS2 obraz kořenového systému souborů

WEB/
        .html, img/*            - zdrojové kódy a obrázky použitých webových stránek
```

Přílohy

1. Zprovoznění prostředí Xilinx ISE Design Suite

Prvním krokem je instalace prostředí, ta není nijak komplikovaná a provedeme ji dle instalační příručky [3]. Dále je nutné provést jisté úpravy, ty jsou převzaty v literatury [8], jsou ale mírně upraveny a především doplněny. Následující text pak počítá s výchozím nastavením instalačního adresáře, který je /opt/Xilinx.

Prvním bodem je zprovoznění ovladačů pro vývojové desky. Bez nich nelze s deskou komunikovat.

To provedeme následujícími příkazy:

```
cp /opt/Xilinx/12.4/ISE_DS/ISE/bin/lin64/xusbdfwu.rules
/etc/udev/rules.d/50-xusbdfwu.rules

sed -i -e 's/TEMPNODE/tempnode/' -e 's/SYSFS/ATTRS/g' -e
's/BUS/SUBSYSTEMS/' /etc/udev/rules.d/50-xusbdfwu.rules

cp /opt/Xilinx/12.2/ISE_DS/ISE/bin/lin64/xusb*.hex /usr/share/

chmod 644 /usr/share/xusb*.hex
```

Dále je třeba doinstalovat do operačního systému aplikace fxload a libusb, které slouží ke komunikaci s usb zařízením z uživatelského prostoru. V distribuci Gentoo tedy příkazem:

```
emerge fxload libusb
```

Dalším krokem je kompilace a instalace knihovny, která zachytává systémová volání USB aplikací ISE, a předává jej výše uvedeným aplikacím v uživatelském prostoru.

1. Stáhneme na stránce http://rmdir.de/~michael/xilinx/ soubor usb-driver-HEAD.tar.gz
2. Rozbalíme soubor `tar -xvzf usb-driver-HEAD.tar.gz` a vstoupíme do vytvořeného adresáře usb-driver-HEAD
3. Spustíme kompilaci příkazem `make`, to vytvoří soubor libusb-driver.so
4. Tento soubor nakopírujeme do instalačního adresáře ISE příkazem: `cp lisbusb-driver.so /opt/Xilinx/12.4/ISE_DS/EDK/lib/lin64/`

Nyní je možné spouštět XPS (z něhož lze spouštět i SDK) pomocí série příkazů:

```
export LC_ALL=en_US.UTF-8
export LD_PRELOAD=/opt/Xilinx/12.4/ISE_DS/EDK/lib/lin64/libusb-
driver.so
source /opt/Xilinx/12.4/ISE_DS/settings64.sh
/opt/Xilinx/12.4/ISE_DS/EDK/bin/lin64/xps
```

První řádek nastaví jazykové prostředí na angličtinu, jiné jazyky jako je čeština mají v OS Linux jiný význam znaků desetinné čárky a tečky což vede při používání k chybám, které znemožní i prosté základní vytvoření návrhu. Druhý řádek pak pomocí systému LD_PRELOAD začne používat dříve zkompilovanou knihovnu libusb-driver.so. Tento systém funguje tak že běžící aplikaci převezme jistá definovaná systémová volání a nahradí je funkcemi z této knihovny. Tento bod umožní nahrávat návrhy do vývojové desky, ovšem způsobuje i problémy, které budou vysvětleny a řešeny dále. Třetí řádek pouze nastaví proměnné prostředí nutné k používání ISE. Poslední řádek pak přímo spustí vývojové prostředí XPS. Tuto posloupnost příkazů je potřeba použít vždy při spuštění v novém shellu. Výhodné je umístit je do nějakého skriptu.

Následně je potřeba upravit TCL skript /opt/Xilinx/12.4/ISE_DS/EDK/data/xmd, neboť použití LD_PRELOAD zde způsobí jeho nefunkčnost, to proto že skript spouští jiné programy, které s LD_PRELOAD nejsou kompatibilní a vracejí chybové hlášení. Toto hlášení TCL skript vyhodnotí jako chybu a ukončí se. Bez následující úpravy tak například v EDK není funkční zápis jádra a souborového systému do paměti Flash.

1. Otevřeme skript textovým editorem a okolo řádku 140 končí definice globálních proměnných a následuje vlastní tělo skriptu.
2. Tuto část skriptu upravíme takto:

```
...
      global DEV_ALGO
      global env

      set env(LD_PRELOAD) " "


;# -------------------- -------------------- --------------------
;# Silence XMD

...
```

Tímto docílíme vymazání proměnné LD_PRELOAD v rámci běhu skriptu. Tak se pro vykonávání skriptu nebude používat a skript nyní proběhne v pořádku.

2. Zdrojový kód zavaděče:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "blconfig.h"
#include "portab.h"

/* Declarations */

static uint8_t copy_to_ram ();
extern void init_stdout();

#ifdef __cplusplus
extern "C" {
#endif

extern void outbyte(char c);

#ifdef __cplusplus
}
#endif



static uint8_t *rambuf;
static uint8_t *flbuf;


/* We don't use interrupts/exceptions.
   Dummy definitions to reduce code size on MicroBlaze */
#ifdef __MICROBLAZE__
void _interrupt_handler () {}
void _exception_handler () {}
void _hw_exception_handler () {}
#endif


int main()
{
    uint8_t ret;

    init_stdout();


    print ("\r\nBIN Bootloader\r\n");
    print ("Loading BIN image from flash @ address: ");
```

```
    putnum (FLASH_IMAGE_BASEADDR);
    print ("\r\n");

    rambuf = (uint8_t*)RAM_ADDR;
    flbuf = (uint8_t*)FLASH_IMAGE_BASEADDR;
    ret = copy_to_ram ();

    return ret;
}


static uint8_t copy_to_ram ()
{

    void (*laddr)();

    memcpy ((void*)rambuf, (void*)flbuf, 0x600000);


     laddr = (void*)rambuf;

    print ("\r\nExecuting Linux kernel at address: ");
    putnum ((uint32_t)laddr);
    print ("\r\n");


    (*laddr)();

    return 0;
}



#ifdef __PPC__

#include <unistd.h>

/* Save some code and data space on PowerPC
   by defining a minimal exit */
void exit (int ret)
{
    _exit (ret);
}
#endif
```